

Aplikační firewall na bázi SIP Proxy Kamailio a SDN

Application-layer Firewall Based on SIP Proxy Kamailio and SDN

Bc. Lukáš Grunert

Diplomová práce

Vedoucí práce: Ing. Jan Rozhon, Ph.D.

Ostrava, 2021

Abstrakt

Tato diplomová práce se zabývá návrhem aplikačního firewallu. Cílem bylo vytvořit funkční řešení firewallu za pomoci Kamailio SIP serveru a SDN infrastruktury. Práce je rozdělena na teoretickou a praktickou část. V teoretické části je představen Kamailio server a jeho části, základy SDN a virtuální přepínač Open vSwitch. V praktické části je detailní popis návrhu a postupu řešení. Výsledné řešení zahrnuje Kamailio, Open vSwitch a Ryu SDN kontrolér. Závěr praktické části je věnován testování a porovnání navrženého řešení s tradičními přístupy.

Klíčová slova

SIP, Kamailio, SDN, Ryu, Open vSwitch, OpenFlow, Aplikační firewall

Abstract

This diploma thesis deals with the design of an application firewall. The goal was to create a functional firewall solution using Kamailio SIP server and SDN infrastructure. The work is divided into theoretical and practical part. The theoretical part introduces the Kamailio server and its parts, the basics of SDN and the virtual switch Open vSwitch. The practical part is a detailed description of the design and solution procedure. The resulting solution includes Kamailio, Open vSwitch and Ryu SDN controller. The conclusion of the practical part is devoted to testing and comparing the proposed solution with traditional approaches.

Keywords

SIP, Kamailio, SDN, Ryu, Open vSwitch, OpenFlow, Application-layer Firewall

Poděkování

Mé poděkování patří Ing. Janu Rozhonovi, Ph.D. za odborné vedení, trpělivost a ochotu, kterou mi v průběhu zpracování diplomové práce věnoval.

Obsah

Seznam použitých zkratek	6
Seznam obrázků	8
Seznam tabulek	9
1 Úvod	12
2 Protokoly	14
2.1 SIP	14
2.2 OpenFlow	16
2.2.1 Multipart Request a Reply	17
2.2.2 Flow Mod	17
2.2.3 Echo Request a Reply	17
3 Kamailio server	19
3.1 Nástroje	19
3.2 Pseudoproměnné	20
3.3 Moduly	20
3.3.1 PIKE	21
3.3.2 Permissions	22
3.3.3 Sanity	23
3.3.4 HTable	24
3.3.5 Geoip2	25
3.3.6 Dialog	26
3.3.7 HTTP_ASYNC_CLIENT	27
3.3.8 JSONRPC-S	28
4 SDN	30
4.1 Kontrolér	32

4.1.1	OpenDaylight	32
4.1.2	ONOS	33
4.1.3	Ryu	34
5	Open vSwitch	38
5.1	Architektura	39
5.2	Nástroje	39
5.3	Flow tabulka	40
6	Praktická část	42
6.1	Návrh zabezpečení	42
6.2	Topologie	43
6.3	Instalace	44
6.4	Konfigurace a testování	46
6.4.1	Nastavení Open vSwitch mostu	46
6.4.2	Ovládání Open vSwitch Flow tabulky skrze Ryu	48
6.4.3	Útok typu Flood	53
6.4.4	Útok typu SQL injekce	55
6.4.5	Skenování sítě	56
6.4.6	VoIP spam	58
6.4.7	SIP zpráva z nepovoleného státu	59
6.4.8	Odhalení pozměněné/poškozené SIP zprávy	61
6.4.9	Pokus o nepovolenou registraci účastníka	62
6.4.10	Počáteční nastavení sítě skrze nástroj kamctl	63
6.5	Shrnutí navrženého řešení	66
7	Závěr	68
	Literatura	69
	Přílohy	71
A	Obrázky	72
B	Kód	74
C	Struktura přiložených souborů	77

Seznam použitých zkratek

AAA	– Authentication, Authorization, Accounting
API	– Application Programming Interface
BGP-LS	– Border Gateway Protocol - Link State
GRE	– Generic Routing Encapsulation
DPID	– Datapath Identifier
HTTP	– Hypertext Transfer Protocol
IP	– Internet Protocol
JSON	– JavaScript Object Notation
JSONRPC	– JavaScript Object Notation Remote Procedure Call
JVM	– Java Virtual Machine
LACP	– Link Aggregation Control Protocol
LISP	– Locator/ID Separation Protocol
NTT	– Nippon Telegraph and Telephone
ODL	– OpenDayLight
ONOS	– Open Network Operating System
OSGi	– Open Services Gateway initiative
OVSDB	– Open vSwitch Database management Protocol
PCEP	– Path Computation Element communication Protocol
QoS	– Quality of Service
RPC	– Remote Procedure Call
SDN	– Software Defined Network
SIP	– Session Initiation Protocol
SNMP	– Simple Network Management Protocol
SPIT	– Spam over Internet Telephony
SQL	– Structured Query Language
STT	– Stateless Transport Tunneling protocol
TCP/IP	– Transmission Control Protocol/Internet Protocol
TL1	– Transaction Language 1

UA	– User Agent
UAC	– User Agent Client
UAS	– User Agent Server
URI	– Uniform Resource Identifier
VLAN	– Virtual Local Area Network
VM	– Virtual Machine
VoIP	– Voice over Internet Protocol
VxLAN	– Virtual extensible Local Area Network
WSGi	– Web Service Gateway Interface
XML	– Extensible Markup Language
YANG	– Yet Another Next Generation

Seznam obrázků

4.1	Srovnání tradiční a SDN sítě	31
4.2	Model zpracování událostí v Ryu prostředí	35
5.1	Hlavní komponenty a nástroje architektury Open vSwitch[30]	40
6.1	Použitá topologie při návrhu firewallu	43
6.2	Nastavení OvS mostu	47
6.3	Ukázka odchycené komunikace mezi Kamailio serverem a Ryu kontrolérem	49
6.4	OpenFlow zpráva Multipart Request	50
6.5	OpenFlow zpráva Multipart Reply	51
6.6	OpenFlow zpráva Flow Mod	52
6.7	Ukázka nastavení hard timeout ve Flow tabulce Open vSwitch přepínače	52
6.8	Ukázka výpisů Kamailio serveru při útoku typu flood	54
6.9	Přihlašovací údaje pro testování útoku typu SQL injekce	56
6.10	Ukázka výpisů Kamailio serveru při útoku typu SQL injekce	56
6.11	Ukázka odpovědi na přihlašování pod správným a špatným uživatelským jménem	56
6.12	Ukázka skenování sítě pomocí nástroje SipVicious	57
6.13	Ukázka výpisů Kamailio serveru při skenování sítě	57
6.14	Ukázka výpisů Kamailio serveru při krátkém hovoru	59
6.15	Ukázka výpisů Kamailio serveru během SIP komunikace z nechtěné země	60
6.16	Ukázka výpisů Kamailio serveru během SIP komunikace s poškozenou SIP hlavičkou	61
6.17	Ukázka SIP odpovědi Kamailio serveru na poškozenou SIP zprávu	62
6.18	Ukázka výpisu Kamailio serveru během registrace ze špatné IP adresy	63
6.19	Počáteční nastavení sítě skrze nástroj kamctl	63
A.1	Porovnání volání příkazu subnetDump vzdáleně a lokálně	72
A.2	OpenFlow Keepalive zprávy	73
A.3	Spouštění Ryu aplikace jako službu v systemd	73
A.4	Vytváření a mazání záznamů z Ryu databáze	73

Seznam tabulek

4.1	Srovnání největších rozdílů SDN platforem OpenDaylight a ONOS	34
4.2	Syntaxe REST příkazů Firewall aplikace	36
C.1	Struktura přiloženého souboru k diplomové práci	77

Seznam kódů

3.1	Nastavení parametrů modulu PIKE	21
3.2	Nastavení spojení do databáze pro funkci allow_source_address modulu Permissions	22
3.3	Nastavení parametru autodrop pro modul Sanity	23
3.4	Vytvoření hash tabulky ipban modulem HTable	24
3.5	Ukázka práce s hash tabulkou modulu HTable	25
3.6	Nahrání GeoLite2 databáze pro účely práce s GeoIP2 modulem	25
3.7	Práce s modulem Geoip2	26
3.8	Měření doby hovoru pomocí modulu Dialog	26
3.9	Nastavení těla HTTP POST zprávy	27
3.10	Zpracování HTTP POST zprávy a odpovědi v Kamailio konfiguračním souboru	28
3.11	Zpracování JSONRPC volání	29
6.1	Instalace Kamailio serveru	44
6.2	Instalace Open vSwitch přepínače	45
6.3	Instalace Ryu frameworku	45
6.4	Přidání Ryu aplikace do systemd	46
6.5	Nastavení Open vSwitch mostu	47
6.6	Nastavení Open vSwitch záznamu pro blokování provozu	51
6.7	Útok typu flood pomocí nástroje inviteflood	53
6.8	Zachycení útoku typu flood a zaslání informace na Ryu kontrolér	53
6.9	Zachycení útoku typu SQL injekce a zaslání informace na Ryu kontrolér	55
6.10	Zachycení skenování sítě a zaslání informace na Ryu kontrolér	56
6.11	Zachycení krátkého hovoru a zaslání informace na Ryu kontrolér	58
6.12	Zachycení SIP zprávy z nežádoucího státu a zaslání informace na Ryu kontrolér	59
6.13	Nastavení pákistánské IP adresy a zaslání SIP zprávy	60
6.14	Zachycení poškozené/pozměněné SIP zprávy a zaslání informace na Ryu kontrolér	61
6.15	Příkaz nástroje SIPp pro testování poškozených SIP zpráv	61
6.16	Příkaz nástroje SIPp pro testování poškozených SIP zpráv	62
6.17	Python metody pro Permissions modul	64
6.18	Začátek metody regist_ofs	65

6.19	Zpracování definovaných sítí a zaslání na OvS přepínač	65
B.1	Zpracování dat z Kamilio serveru v Ryu aplikaci	74

Kapitola 1

Úvod

Mimo rozmach nových technologií zažily v posledních letech komunikační sítě také zájem o nové paradigmatata či přístupy k tomu, jak sítě navrhovat a spravovat. Jeden z nejmodernějších přístupů posledních deseti let je tzv. Softwarově definovaná síť, ve které hraje velkou roli centralizace správy, kterou obstarává síťové zařízení pod obecným názvem kontrolér, a programové řízení, což je soubor aplikací představujících inteligenci sítě. Skrze tento přístup lze vytvořit takové síťové funkce, které v tradičních sítích realizovány být nemohou, či jen s obtížemi. Práce se zabývá návrhem aplikačního firewallu, který je na zmíněném přístupu softwarově definovaných sítí založen.

Práce je rozdělena do dvou hlavních částí - teoretické a praktické. První kapitola teoretické části se věnuje stěžejním protokolům této práce - SIP a OpenFlow. V druhé kapitole je popsán SIP server Kamailio. Ten v návrhu firewallu funguje jako prvek, jenž odhaluje bezpečnostní hrozby nad protokolem SIP. Při popisu je kladen důraz na seznámení čtenáře s moduly Kamailia, které pro odhalování hrozeb slouží. Následuje kapitola popisující základy moderního síťového paradigmatu Softwarově definovaných sítí, v rámci které jsou popsány tři mnou vybrané open-source SDN kontroléry. Největší důraz je kladen na Ryu kontrolér, jelikož je použit v praktické části. Poslední kapitola teoretické části se věnuje virtuálnímu přepínači Open vSwitch. Ten ve výsledném řešení zastává funkci firewallu.

V rámci této práce není možné popsat, zachytit a blokovat všechny bezpečnostní hrozby, které by se při používání SIP protokolu mohly vyskytnout. Je třeba si také uvědomit, že návrh zabezpečení sítě je závislý na konkrétní architektuře sítě a využívaných službách. Firewall je příklad použití SIP serveru Kamailio a SDN. Výsledné řešení slouží pouze jako příklad, který lze dále upravovat. Na začátku praktické části je sekce představující seznam bezpečnostních hrozeb, které firewall dokáže identifikovat, zpracovat a blokovat. Následují sekce s popisem použité síťové topologie, s instalacemi jednotlivých síťových entit, konfiguracemi a testováním. V poslední kapitole je porovnání výsledného řešení s tradičními přístupy.

Jelikož je v praktické části využito několik protokolů, principů zasílání dat, a scénářů s útoky na SIP server Kamailio, je vhodné v rámci práce uvést odchycení síťových komunikací. Dále je potřeba předvést a vysvětlit klíčové kusy kódů. V praktické části je uvedena a vysvětlena většina obrázků a kódů. V příloze A jsou doplňující obrázky, v příloze B doplňující úryvek kódu Ryu aplikace, a v příloze C je popsána struktura přiložených souborů k práci.

Kapitola 2

Protokoly

Tato kapitola popisuje dva protokoly - SIP a OpenFlow. Cílem práce je vytvořit aplikační firewall, což znamená, že bude blokovat provoz na základě hrozeb nalezených v aplikační vrstvě modelu TCP/IP. Aplikačním protokolem, nad kterým firewall provádí kontrolu, je signální protokol SIP. Dalším cílem práce je využít moderního síťového paradigmatu Softwarově definovaných sítí. Toto paradigma používá model správy síťových zařízení skrze centralizovaný prvek - kontrolér. Jedním z protokolů, které se v SDN používají ke správě zařízení, je OpenFlow. Tento protokol je praktické části práce.

2.1 SIP

SIP je signální protokolem, který operuje na aplikační vrstvě TCP/IP modelu. Základy první verze byly popsány v RFC 2543. Současná verze SIP 2.0 je popsána v RFC 3261. SIP je univerzálním protokolem pro vytvoření, správu a ukončení komunikačních relací mezi dvěma a více účastníky. Mezi základní vlastnosti patří přidání účastníka do již existující relace, přidávání a odebrání médií v rámci již existující relace, nebo služba přesměrování a prezenze. Funguje nezávisle na základních transportních protokolech a typu nastavované relace. Je velmi často nasazován v multimediálních systémech, konkrétně ve VoIP. Běžná komunikace nejčastěji funguje nad UDP s portem 5060, ta šifrovaná pak nad TLS s portem 5061. SIP si dává za cíl být co nejjednodušší a nechává se inspirovat tím, co v internetu dobře funguje. Z tohoto důvodu je velmi podobný protokolu HTTP. SIP je textově orientovaný protokol (s UTF-8 kódováním) a je postaven na modelu klient-server. Klient je v SIP terminologii nazýván UA (User Agent). K adresaci je v SIP využita tzv. SIP URI, která má obecný formát: sip:název_úctu@doména:port.[1]

Komunikace mezi SIP entitami probíhá pomocí žádostí a odpovědí. Klient zasílá žádost serveru, server mu v reakci posílá odpověď. Následující seznam popisuje šest základních SIP žádostí:[1]

- **REGISTER** - registrace účastníka

- **INVITE** - zahájení relace
- **ACK** - potvrzení zahájení relace
- **CANCEL** - ukončení zahajované relace
- **BYE** - ukončení již vzniklé relace
- **OPTIONS** - dotazování se na technické možnosti

Odpovědi na žádosti obsahují třímístný stavový kód. První číslice kódu určuje třídu odpovědi, pomocí zbylých dvou číslic lze pak konkretizovat důvod odpovědi. Současná verze protokolu SIP definuje 6 tříd odpovědí:[1]

- **1xx** - informační zprávy - žádost byla přijata a zpracovává se
- **2xx** - žádost byla úspěšně přijata a vyřízena
- **3xx** - zpráva o přesměrování, k úspěšnému vyřízení žádosti budou potřeba dodatečné kroky
- **4xx** - žádost nemůže být zpracována z důvodu špatné syntaxe, nebo dotazovaný server nemá naimplementovanou požadovanou službu
- **5xx** - žádost nemůže být zpracována z důvodu chyby serveru
- **6xx** - žádost nemůže být vyřízena na žádném serveru

Odpovědi z třídy 1xx jsou jediné, kterých může v reakci na žádost přijít více než jedna. U zbylých pěti tříd funguje princip zaslání vždy jedné odpovědi pro jednu žádost - označují se jako odpovědi konečné.

Hlavička SIP zprávy je textově orientovaná a je velmi podobná hlavičce HTTP zprávy. Jednotlivá pole mají tvar *název_pole:hodnota_pole*. RFC 3261 definuje šest základních polí, které musí být v SIP zprávě přítomny vždy a pomocí kterých se řeší většina nejkritičtějších operací. Za tato pole se mohou přidávat pole dodatečná. Základní pole jsou tato:

- **Via** - označuje transportní protokol a identifikuje místo, kam má být zaslána odpověď na žádost; typický vzhled je Via: SIP/2.0/UDP 10.11.12.13:5060
- **Max-forwards** - omezení počtu přeskoků při cestě k cíli, odesílatel by měl hodnotu nastavit na hodnotu 70 a každá následující SIP entita pak hodnotu tohoto pole sníží o 1, při hodnotě 0 se zpráva zahazuje a odesílatel obdrží chybovou odpověď 483 (Too Many Hops)
- **Call-ID** - je unikátní identifikátor SIP dialogu - uskupení všech žádostí a odpovědí v rámci relace; typický vzhled je Call-ID: f81d4fae-7dec-11d0-a765-00a0c91e6bf6@example.com

- **CSeq** - je unikátní identifikátor SIP transakce uvnitř dialogu; typický vzhled je CSeq: 4711 INVITE
- **From** - definuje logickou identitu odesílatele žádosti, musí obsahovat tag; typický vzhled je From: User1 <sip:user1@example.com>;tag=ab12
- **To** - definuje logickou identitu příjemce žádosti, který může a nemusí být konečným příjemcem; typický vzhled je To: User1 <sip:user1@example.com>

Dále existují pole, která sice nejsou povinná, avšak jsou hojně využívána - např. Contact (s přímou cestou ke kontaktování odesílatele požadavku), Content-Type (uvádí povahu těla SIP zprávy) a Content-Length (s počtem Bytů v těle zprávy). Hlavička SIP zprávy je pak zakončena prázdným řádkem, označovaným jako CLRF.[1]

2.2 OpenFlow

OpenFlow je komunikační protokol, jehož hlavním účelem je správa přepínacích tabulek síťových zařízení skrze kontrolér. Základ protokolu vznikl na Stanfordské univerzitě, a to mezi lety 2006 až 2011. Od roku 2011 se o vývoj a standardizaci protokolu stará nezisková organizace Open Networking Foundation, jejíž členové jsou např. Google, AT&T nebo Deutsche Telekom. OpenFlow kontrolér by měl poslouchat na portu 6653, popř. 6633. OpenFlow využívá TCP pro nezabezpečené a TLS pro zabezpečené spojení. Nejnovější verze protokolu je 1.5.1. [32]

Aby mohl OpenFlow kontrolér síťová zařízení spravovat, potřebuje s nimi mít navázané TCP nebo TLS spojení. Na základě SDN aplikací pak může provádět úkony s Flow tabulkou, jako např. vytvoření, pozměnění či odebrání flow záznamu, nebo také sběr statistik. Podle využívání se OpenFlow dělí na dvě části:

- **OpenFlow wire protokol** - slouží k navázání spojení, definování flow záznamu, popř. smazání či změně tohoto záznamu, a ke sběru statistik
- **OF-config** - poskytuje přímý přístup ke konfiguraci zařízení, např. nastavení IP adresy či portu

V práci je využit pouze OpenFlow wire protokol.

Specifikace OpenFlow zahrnuje popis jednotlivých zpráv, portů, tabulek, skupin, čítačů, instrukcí a akcí. V příštích třech sekcích je popsáno pouze pět typů konkrétních OpenFlow zpráv, které jsou v práci používány nejčastěji. Multipart Request a Reply Ryu kontroléru slouží pro získávání záznamu z Flow tabulky přepínače, Flow Mod pro nastavení nového záznamu a zprávy Echo Request a Reply pro udržení spojení a zjišťování, zda je protějščí člen komunikace stále dostupný. Celá specifikace protokolu OpenFlow je k nalezení v [32].

2.2.1 Multipart Request a Reply

Multipart Request a Reply zprávy patří k vícedílným zprávám protokolu OpenFlow, které se posílají mezi kontrolérem a ovládanými přepínači. Žádost Multipart Request kontroléru slouží pro dotazování se přepínače na jeho stav, záznamy ve Flow tabulce, statistiky apod. Přepínač definovaná data zasílá zpět kontroléru jako Multipart Reply odpověď. Aby se OpenFlow zpráva považovala za Multipart Request, musí mít pole Type v OpenFlow hlavičce hodnotu 18. Multipart Reply pak používá hodnotu 19.[32]

Oba typy zpráv rozšiřují originální OpenFlow hlavičku o další pole, mezi kterými je i druhé pole Type. V tomto poli se konkretizuje druh informace, kterou má přepínač kontroléru posílat. V praktické části práce se velmi často pracuje s nastavením druhého pole Type na hodnotu 1, což zprávu interpretuje do OFPMP FLOW zprávy. Ta kontroléru slouží jako žádost o záznam z Flow tabulky přepínače. V rámci práce se kontrolér doptává na všechny přítomné záznamy, a to pomocí tzv. ALL a ANY hodnot - OFPTT ALL pro označení všech tabulek, OFPP ANY pro všechny porty a OFPG ANY pro všechny skupiny. Jako odpověď přijde skoro vždy více než jeden záznam. Tělo odpovědi je pak často polem záznamů. Informace o jednotlivých záznamech obsahují např. číslo flow tabulky, dobu záznamu v tabulce, prioritu, idle a hard časovače, cookie, akce a statistiky.[32] Ukázka použití Multipart Request a Reply zpráv je v sekci 6.4.2.

2.2.2 Flow Mod

OpenFlow zpráva Flow Mod slouží k modifikaci Flow tabulky. Uvnitř zprávy lze definovat, zda bude přidán, odebrán nebo pozměněn záznam. Aby se OpenFlow zpráva považovala za Flow Mod, musí mít pole Type v OpenFlow hlavičce hodnotu 14. Flow Mod definuje pět příkazů: ADD (k přidání záznamu), MODIFY, resp. MODIFY STRICT (ke změně záznamu) a DELETE, resp. DELETE STRICT (ke smazání záznamu). STRICT příkazy definují přesný záznam, a to pomocí polí cookie a priority. Např. příkaz DELETE pro konkrétní zdrojovou a cílovou IP adresu by smazal všechny záznamy, které by splňovaly podmínku IP adres. U STRICT verze DELETE by se pomocí priority a cookie nadefinoval konkrétní záznam, který se má smazat.

V práci je nejčastěji využit příkaz ADD. Uvnitř tohoto příkazu jsou definovány pole, které se do záznamu mají přidat, např. zdrojová a cílová IP adresa, idle a hard časovače, priorita, číslo flow tabulky, port a instrukce.[32] Ukázka použití Flow Mod zprávy je v sekci 6.4.2.

2.2.3 Echo Request a Reply

Žádost Echo Request a odpověď Echo Reply slouží ke kontrole dostupnosti protější strany komunikace, k udržení spojení, a k zasílání dat pro účely měření. Zprávy jsou sestaveny z OpenFlow hlavičky a těla zprávy, které může mít libovolnou délku. Tělo odpovědi by mělo být kopií těla

žádosti. Specifikace uvádí, že se v těle mohou posílat časová razítka pro potřebu měření síťového zpoždění, data různých velikostí pro potřeby měření přenosového pásma, nebo nulová výplň pro účely udržování spojení mezi kontrolérem a spravovaným zařízením. Využití zpráv k udržení spojení se souhrnně označuje jako zasílání Keepalive zpráv. V práci je zasílání Keepalive zpráv využito, a to s periodou zasílání 5 sekund.[32]

Kapitola 3

Kamailio server

Kamailio je open-source SIP server, jenž je schopen sestavovat tisíce telefonních hovorů za sekundu. Vzhledem ke svému vysokému výkonu a velkému počtu dodatečných modulů, které rozšiřují jeho schopnosti, má Kamailio potenciál k vytváření velkých VoIP infrastruktur. Jedná se o tzv. event-based (událostmi řízený) server a veškerá logika zpracování událostí (zejména SIP požadavků) je konfigurována uvnitř hlavního konfiguračního souboru - **kamailio.cfg**. Tento soubor využívá svůj vlastní jazyk. Do hlavního konfiguračního souboru lze přidat dodatečné funkcionality skrze nahrání tzv. modulů. Podkapitola, která se zabývá použitými moduly v této práci, je 3.3. Kamailio má po celou dobu své existence aktivní vývojářskou komunitu a rozvíjející se webové stránky s návody a komentáři. Ke komunikaci s vývojáři, obecné diskusi, vznášení požadavků, získávání nejnovějších informací slouží Kamailio Mailing list. Kamailio vývojáři také využívají Development tracker, který je na platformě GitHub.[14][15]

V době psaní této práce je nejnovější verze serveru Kamailio 5.4.4. Na oficiálních stránkách Kamailio Wiki jsou ke každé verzi k dispozici podstránky, které obsahují informace o konfiguraci jádra, o pseudoproměnných, transformacích, výběrech, instalaci, tutoriálech a mnoho dalšího. Konfigurace v rámci této práce se v drtivé většině spoléhá na podstránky o konfiguraci jádra, pseudoproměnných a konkrétních modulech, které byly použity (viz 3.3).[14][15]

3.1 Nástroje

Kamailio server má k dispozici pět nástrojů pro účely správy serveru - **kamctl**, **kamdbctl**, **kamcmd**, **siremis** a **kamcli**. V této práci jsou použity první tři nástroje:[13]

- **kamctl** - je instalován automaticky instalován při instalaci serveru. Má vlastní konfigurační soubor - *kamctlrc* - uložený ve stejné složce, jako hlavní konfigurační soubor (*kamailio.cfg*). Nachází se zde nastavení SIP domény, přihlašovací údaje k databázi, nastavení databázového

serveru apod. Nástroj jako takový slouží zejména k přidávání/správu/odstraňování SIP účtů a ke kontrole registrovaných účtů a statistik serveru

- **kamdbctl** - je instalován společně s Kamilio serverem. Nástroj používá stejný konfigurační soubor, jako nástroj kamctl. Používá se ke komunikaci s databázovým serverem. Definuje řadu skriptů, např. pro vytvoření kamilio databáze. V případě, že bude Kamilio využívat databázový backend, by měl tento nástroj být použit k vytvoření databáze neprodleně po instalaci serveru
- **kamcmd** - je také instalován společně s Kamilio serverem. Tento nástroj je určen k posílání RPC příkazů do Kamilio serveru, a to skrze příkazový řádek. Pro fungování je potřeba mít v Kamilio konfiguračním souboru nahrán modul ctl.[13]

3.2 Pseudoproměnné

Obecnou vlastností pseudoproměnných je reference na konkrétní prvek, nejčastěji pole v SIP hlavičce (např. \$ci pro Call-ID). Uplatňují se však také k referenci na některé systémové proměnné (např. \$Tb pro čas zapnutí Kamilia), referenci do jiných protokolů (např. \$http_rb pro tělo HTTP odpovědi), a nebo také jako speciální reference některých modulů (např. \$sht(htable=>key) pro odkaz na položku v hashovací tabulce HTable modulu). Každá pseudoproměnná začíná symbolem \$. Většinu pseudoproměnných lze použít po nahrání modulu pv (pseudo-variable modul). Zbýlé pseudoproměnné jsou vždy vázány na konkrétní modul, a jsou použitelné vždy až po úspěšném nahrání modulu.[12]

3.3 Moduly

Moduly slouží k dodání nadstandardních funkcionalit do konfiguračního jazyka Kamilia. Pro Kamilio verze 5.4.x, které je používáno v této práci, existuje přes 230 tzv. stabilních modulů. V této kapitole jsou popsány jen ty moduly, které jsou pro tuto práci klíčové. Pro potřeby zachycení bezpečnostních hrozeb a útoků jsou využity moduly PIKE, Permissions, Sanity, HTable, Geoip2 a Dialog. Pro potřeby serverové komunikace mezi Kamiliem a Ryu kontrolérem jsou použity moduly HTTP_ASYNC_CLIENT a JSONRPC-S. Po nainstalování Kamilio serveru je v konfiguračním souboru již předpřipravená základní konfigurace a nahrávají se zde výchozí moduly. Ty jsou ponechány v základním nastavení a dále v této kapitole popisovány nebudou. Většina modulů obsahuje popis parametrů, funkcí, pseudoproměnných a RPC příkazů. Parametry slouží k nastavení modulu. Skrze funkce se v kódu hlavního konfiguračního souboru implementuje to, k čemu daný modul slouží. Pseudoproměnné se odkazují na data v rámci modulu. RPC příkazy obvykle slouží ke kontrole.

3.3.1 PIKE

Tento modul slouží k nastavení limitace počtu příchozích SIP požadavků za jednotku času, a k detekci překročení tohoto limitu. Jako zdroj požadavků, nad kterým modul počítá překročení limitu, je zdrojová IP adresa. Modul dokáže pracovat zároveň s IP verze 4 i 6. Po překročení zadaného limitu modul zdrojovou IP adresu označí jako blokovanou, což doprovází výpis do terminálu. Ve skutečnosti však IP adresa blokována není. Logiku blokace si musí správce Kamailio serveru zadefinovat explicitně sám, a to např. pomocí modulu HTable (viz 3.3.4). PIKE modul není závislý na žádných dalších modulech či knihovnách.[3]

PIKE definuje 4 parametry. V práci jsou využity dva.

```
1 modparam("pike", "reqs_density_per_unit", 30)
2 modparam("pike", "sampling_time_unit", 2)
```

Kód 3.1: Nastavení parametrů modulu PIKE

První parametr definuje počet požadavků, druhý parametr pak časovou jednotku. Dohromady utvářejí celek, který by se dal interpretovat jako „limit, po jehož překročení bude zdrojová IP adresa blokována, se rovná reqs_density_per_unit požadavků za dobu měření sampling_time_unit“. Autoři modulu však upozorňují, že počet požadavků není přesný, a je v intervalu reqs_density_per_unit až 3*reqs_density_per_unit pro IP verze 4 a reqs_density_per_unit až 8*reqs_density_per_unit pro IP verze 6. Základní nastavení limitace je 30 požadavků za 2 sekundy. U nastavení sampling_time_unit autoři modulu také upozorňují, že při nízké hodnotě může způsobit snížení výkonu serveru, jelikož bude přehlcen proces časovače. Pro detekci skokového nárůstu požadavků (např. při útoku typu flood) je však nutné mít tuto hodnotu co nejnižší. Příliš nízké hodnoty obou parametrů by však mohly způsobit blokaci běžné SIP komunikace.

Co se funkcí týče, modul PIKE má pouze jednu - pike_check_req(). Pokud je v konfiguračním souboru zavolána, zpracuje zdrojovou IP adresu příchozího požadavku a vrátí jednu ze tří numerických hodnot - 1, -1 či -2. Jednička je pro případ, že zdrojová IP adresa nepřekročila stanovený limit pro blokaci. Tvůrci modulu nastavili modul tak, aby vracel jedničku také pro případ vnitřní chyby serveru, aby nedošlo k blokaci nevinného SIP účastníka. Míňusové hodnoty funkce vrátí, pokud zdrojová IP adresa limit překročila poprvé (-2), nebo vícekrát (-1). V návrhu firewallu je využit případ prvotního zjištění překročení limitu. V tomto případě je zaslána informace na Ryu kontrolér, aby byla zdrojová IP adresa blokována na Open vSwitch přepínači. Mezi odesláním zprávy na Ryu a faktickým blokováním IP adresy na Open vSwitch přepínači proběhne nějaký čas, který bude záviset na vytížení sítě, serverů, použité topologii, počtu přeskoků mezi servery apod. Aby v tomto mezičase bylo omezeno zpracování SIP požadavků z problémové IP adresy, je umístěna do hashovací

tabulky modulu HTable a nově přichází požadavky jsou, po rychlém prohledání této tabulky, ihned zahazovány. Praktická ukázka použití modulu PIKE je v 6.4.3.[3]

3.3.2 Permissions

Tento modul přidává možnost vytváření skupin IP adres, resp. podsítí. S těmito skupinami lze dále pracovat, např. povolit jedné skupině možnost registrace, jiné skupině povolit přístup k RPC příkazům apod., což svou funkcí odpovídá tradičnímu přístupovému listu (ACL). Mimo zpracování zdrojové, resp. cílové IP adresy modul zvládá také práci se SIP URI. Práce se SIP URI v rámci modulu Permissions však v návrhu firewallu využita není. Modul není závislý na jiných modulech či knihovnách, kromě případu, že budou data ukládána v databázi. Pak je potřeba definovat spojení s touto databází - např. skrze modul `db_mysql`. Existuje více přístupu k tomu, kde skupiny spravovat, a kde je nahrávat. Modul umí využít lokální i vzdálenou databázi, či soubory s jasně definovaným formátem (přípony `.allow`, `.deny` a `.list`). V práci je použit přístup, kde jsou jednotlivé IP adresy a podsítě definovány skrze volání RPC příkazů nástrojem `kamctl`. Zadefinované údaje jsou uloženy v lokálním MySQL serveru, přičemž je ponecháno základní nastavení, tedy databáze *kamailio* a tabulka *address*. [4]

Modul definuje 7 funkcí. V práci je využita pouze jedna. Ta dokáže obstarat funkcionalitu na úrovni přístupového listu. Onou funkcí je `allow_source_address()`. Funkci lze použít s parametrem, ale také bez, a to s rozdílnými výsledky. V práci je použita pouze varianta s parametrem. Tento parametr udává číslo skupiny, ve které jsou uloženy IP adresy, resp. podsítě. V dokumentaci modulu je popsáno, že tento způsob volání funkce je ekvivalentní volání `allow_address` s parametry `číslo_skupiny`, `zdrojová_IP` a `zdrojový_port`. Funkce po zavolání vrátí hodnotu `true`, pokud jsou zdrojová IP adresa a zdrojový port právě zpracovávaného SIP požadavku nalezeny v hledané skupině.

K vytvoření a naplnění skupin IP adres je v práci využit nástroj `kamctl`. K uložení těchto údajů MySQL databáze. Pomocí příkazu `kamctl address add <číslo_skupiny> <ip_adresa> <maska> <port> <značka>` se vytvoří záznam, který přidá konkrétní IP adresu, masku a port do konkrétní skupiny. K tomuto záznamu se také přidá značka, což je proměnná ve formátu textového řetězce, kterým lze záznam okomentovat. Pro kontrolu zadaných prvků v databázi pak slouží příkaz `kamctl address show`. Jelikož ale funkce `allow_source_address` porovnává zdrojovou IP a port se skupinou, která je nahrána v mezipaměti Kamailio serveru, musí se v rámci modulu zadefinovat cesta k databázi. Dále je nutné nastavit vložení obsahu databáze do mezipaměti serveru.

```
1 modparam("permissions", "db_url", "mysql://kamailio:kamailiorw@localhost/  
   kamailio")  
2 modparam("permissions", "db_mode", 1)
```

Kód 3.2: Nastavení spojení do databáze pro funkci `allow_source_address` modulu Permissions

K tomu slouží řádky 1 a 2 kódu 3.2. Pomocí příkazu *kamcmd permissions.addressReload* se vloží obsah tabulky *address* databáze *kamailio* do mezipaměti Kamailio serveru. Pro kontrolu záznamů, které byly nahrány do mezipaměti, slouží příkaz *kamctl address dump*. Praktické ukázky použití modulu Permissions lze najít v 3.3.8, 6.4.9 či 6.4.10. [4]

3.3.3 Sanity

Pomocí modulu Sanity lze kontrolovat, zda jsou příchozí SIP zprávy ve špatném formátu, nebo jsou nějakým způsobem poškozeny či pozměněny. Při manipulaci s modulem lze pomocí parametrů nastavit, co přesně se má kontrolovat. Funkcemi se pak v konfiguračním kódu Kamailia určí místa a případy, kdy se má kontrola provádět. Autor modulu upozorňuje, že jelikož každá „sanity“ kontrola vyžaduje nadstandartní výkon v podobě parsování a vyhodnocování, je potřeba si dobře rozmyslet, co a proč se bude kontrolovat. Modul je závislý na modulu *sl* - Stateless replies, a je potřeba, aby byl nahrán před nahráním Sanity modulu.[5]

V práci je nastavován jeden parametr, a využita jedna funkce:

```
1 modparam("sanity", "autodrop", 0)
```

Kód 3.3: Nastavení parametru autodrop pro modul Sanity

Tento parametr určuje, zda modul automaticky zruší zpracování SIP žádosti v případě, že pověřená funkce odhalí problém s příchozí SIP žádostí. Parametr má ve výchozím nastavení hodnotu 1, což může být pro správce serveru výhodné, jelikož se o vše postará modul sám (zpracování SIP žádosti se automaticky ukončí a přejde se na další SIP požadavek v pořadí). V rámci této práce je však nutné informovat Ryu kontrolér o tom, že taková situace nastala, a proto je parametr nastaven na 0.

V práci je využita pouze jedna funkce modulu Sanity, a to *sanity_check*. Funkce může být bez parametrů, nebo mít jeden, který je ve formě číselné hodnoty, což definuje typ kontroly. Ve výchozím stavu funkce ukončí zpracování SIP požadavku, pokud objeví problém při jedné z nastavených kontrol. Jelikož je ale autodrop parametr nastaven na 0, funkce bude při výskytu problému vracet hodnotu -1, což lze využít v podmínce. Pokud je funkce volána bez parametru, provedou se ty kontroly, které jsou definovány v parametru *default_checks*. Pokud tento parametr nemá definováno, co se má kontrolovat, použije se výchozí hodnota kontroly, která představuje číslo 3047. Jednotlivé kontroly modulu jsou označeny mocninami čísla 2. Nejvyšší možné číslo je 16384 (2^{14}). Výchozí hodnota 3047 je tedy součtem čísel 1, 2, 4, 32, 64, 128, 256, 512 a 2048. V práci je využita právě tato hodnota, kontroly v ní budou tedy představeny blíže:

- **1** - kontrola RURI SIP verze (jediná povolená verze je momentálně 2.0)
- **2** - RURI schéma podporované Kamailiem (sip, sips, tel, tels, urn)

- **4** - kontrola přítomnosti základních polí SIP hlavičky (To, From, CSeq, Call-ID, Via)
- **32** - porovnání hodnoty CSeq v rámci transakce
- **64** - kontrola hodnoty CSeq, zda je ve formě celého čísla bez znaménka
- **128** - porovnání velikosti těla zprávy s polem Content-Length
- **256** - kontrola, zda má pole Expires hodnotu celého kladného čísla
- **512** - kontrola, zda Kamailio podporuje všechny položky v poli Proxy-Require
- **2048** - kontrola správnosti formátu hodnot polí pro Digest autentikaci

Praktická ukázka práce s tímto modulem je v sekci 6.4.8.[5]

3.3.4 HTable

HTable modul do konfiguračního jazyka Kamailia přidává možnost vytvářet a spravovat hash tabulky. Hash tabulka je datová struktura, která je optimalizována k velmi rychlému vyhledávání. Údaje jsou ukládány formou klíč/hodnota. Jako klíč může být použito cokoliv, ale mělo by se dodržet pravidlo jednoho typu klíčů v rámci tabulky. V práci je jako klíč použita zdrojová IP adresa SIP požadavku. Hodnoty (nejčastěji číselné) se pak dopočítávají skrze hash funkci (algoritmus), a mapují se ke klíči, ze kterého se hodnota počítala. V Kamailiu je hash tabulka uložena ve sdílené paměti, a přistupovat do ní lze skrze pseudoproměnnou - \$sht. Modul podporuje vytvoření hash tabulky s velikostí od 2^2 do 2^{31} možných záznamů. Dále pak nahrání dat z databáze při startu Kamailio serveru. Modul není závislý na žádných jiných modulech, kromě případu, kdy je povoleno DMQ, v tom případě musí být DMQ modul nahrán před HTable modulem. Knihovny nejsou požadovány žádné.[6][10]

V práci je modul využíván spolu s již zmíněným modulem PIKE, který slouží pro odhalení útoku typu flood. Než je informace o útočnickovi zaslána a zpracována, mohou být na Kamailio zaslány stovky dalších falešných zpráv. Aby se Kamailio alespoň trochu zbavilo zátěže ve formě zpracovávání těchto zpráv, vloží se zdrojová IP adresa útočníka do hash tabulky, a při dalším výskytu zprávy z této IP adresy jí Kamailio jednoduše ihned zahodí. Zde je právě využita schopnost velmi rychlého vyhledávání. Toto rychlé vyhledávání je během útoku typu flood velmi žádoucí. Jelikož je v práci hash tabulka používána pro ukládání IP adres, je pojmenována „ipban“:

```
1 modparam("htable", "htable", "ipban=>size=10;autoexpire=20;")
```

Kód 3.4: Vytvoření hash tabulky ipban modulem HTable

Parametr `size` udává velikost tabulky. Desítka znamená, že `ipban` může pojmout až 2^{10} záznamů. Tato hodnota je zavádějící, jelikož je tabulka implementována tak, že každý záznam je ve skutečnosti seznam. Místo v tabulce tak může dojít až v případě, že dojde místo ve sdílené paměti. Parametr `autoexpire` udává, po kolika sekundách bude záznam z tabulky automaticky vymazán. Nastavení 20 sekund koresponduje se základním nastavením proměnné `lifetime` v Ryu aplikaci.

Práce s hash tabulkou pak může vypadat takto:

```
1 $sht(ipban=>$si) = 1;
2 if($sht(ipban=>$si) != $null) {};
```

Kód 3.5: Ukázka práce s hash tabulkou modulu `HTable`

Na řádku jedna se odehrává vyhledání záznamu v hash tabulce `ipban`, a to na základě zdrojové IP adresy (`$si`). Pokud se záznam najde, nastaví se mu hodnota na číslo 1. Pokud se nenajde, vytvoří se nový záznam s hodnotou 1. Na řádku dva je podmínka, uvnitř které je opět vyhledávání v hash tabulce. Hodnota záznamu se porovnává s hodnotou `null`. V případě, že byl záznam se zdrojovou IP do tabulky v minulých 20 sekundách vložen, a byla mu při tom přidělena hodnota 1, bude se hodnota záznamu lišit od hodnoty `null`. V takovém případě se jedná o útočníka, a SIP požadavek může být zahozen.[6]

3.3.5 Geoip2

Modul `Geoip2` umožňuje nahrát `MaxMind GeoIP2/GeoLite2` databázi do sdílené mezipaměti `Kamailia`. Následně lze porovnávat konkrétní IP adresy s touto databází, a to v reálném čase z konfiguračního souboru. Tato databáze mapuje IP adresy s národním geografickým prostředím. V případě shody funkce modulu vrací informace o původu IP adresy. Informace mohou obsahovat kód země, název města, poštovní kód, časovou zónu atp. Modul není závislý na žádných dalších modulech, potřebuje však, aby byla nainstalována knihovna `libmaxminddb`. Dále je nutné, aby byla ze stránek společnosti `MaxMind` stažena databáze. Existují dva druhy databází, které lze v rámci tohoto modulu použít - `GeoIP2` a `GeoLite2`. Největší rozdíl je ten, že první jmenovaná je placená a přesnější. Obě databáze jsou aktualizovány týdně a mají stejný formát. V práci je použita neplacená databáze, tedy verze `GeoLite2`. [7][11]

V rámci práce s tímto modulem je využit jeden parametr - `path`:

```
1 modparam("geoip2", "path", "/usr/local/etc/kamailio/GeoLite2-City.mmdb")
```

Kód 3.6: Nahrání `GeoLite2` databáze pro účely práce s `GeoIP2` modulem

Tímto parametrem se nastavuje absolutní systémová cesta ke stažené databázi.

Modul definuje pouze jednu funkci - *geoip2_match()*. Tato funkce porovnává vloženou IP adresu s databází v mezipaměti. Zároveň výsledek porovnání vloží do pvc kontejneru, do které lze dále přistupovat pomocí pseudoproměnné:

```
1  if(geoip2_match("$si", "src")){
2      xlog("IP: $si");
3      xlog("CC: $gip2(src=>cc)");
4      xlog("City: $gip2(src=>city)");
5      xlog("Zip Code: $gip2(src=>zip)");
6  }
```

Kód 3.7: Práce s modulem Geoip2

Na prvním řádku kódu 3.7 se porovnává zdrojovou IP adresu s databází. Výsledek se ukládá do kontejneru, který byl pojmenován *src*. Pokud se záznam pro IP adresu v databázi najde, podmínka se splní. V takovém případě se vypíše kód státu, město a poštovní směrovací číslo. Pokud se shoda nenajde, podmínka se nesplní. Přístup do kontejneru *src* je uskutečněn skrze pseudoproměnnou *\$gip2*. Praktická ukázka práce s modulem Geoip2 je v 6.4.7.[7][12]

3.3.6 Dialog

Dialog modul se používá ke správě některých vlastností právě probíhajících dialogů, a ke sběru informací o těchto dialozích. Jedná se o jeden z nejkompexnějších a nejobsáhlejších modulů Kamailio serveru. Po nahrání a vhodné konfiguraci tohoto modulu se Kamailio může chovat jako stavový proxy server. Další možný kandidát, který z Kamailia dělá stavovou proxy, je transaction modul, avšak ten operuje nad transakcemi, ne dialogy. Dialog modul je v práci použit pro měření doby hovoru, a to za účelem odhalení potenciálního zdroje VoIP spamu. Měření probíhá triviálně, doba hovoru je výsledkem měření času mezi metodou INVITE a metodou BYE. V práci nejsou řešeny další aspekty, jako třeba NAT, přesměrování apod. Pro účely ukázky funkčnosti firewallu a testování to však plně stačí. Modul je závislý na modulech TM (Transaction modul), RR (Record-Route modul) a PV (Pseudo-variables modul). Tyto moduly musí být nahrány před Dialog modulem.

Zavoláním funkce *dlg_manage()* začne Kamailio zpracovávat SIP požadavek Dialog modulem. Jedná se o alternativní přístup k nastavení příznaku dialog při prvotním INVITE požadavku a spouštění funkcí při zpracovávání dodatečných požadavků uvnitř probíhajícího dialogu. Samotný přístup k délce hovoru se pak provádí skrze pseudoproměnnou *\$DLG_lifetime*. Tato pseudoproměnná odkazuje na délku dialogu. Hodnota je vypočítána jako interval mezi voláním funkce *dlg_manage()* a aktuálním časem volání této pseudoproměnné.

```
1  if (is_method("INVITE")) {
2      dlg_manage();
```

```

3  }
4  /*
5  code
6  */
7  if (is_method("BYE")) {
8      if ($DLG_lifetime < 10) {
9          xlog("Call duration under 10 seconds from $si:$sp");
10     }
11 }

```

Kód 3.8: Měření doby hovoru pomocí modulu Dialog

Na řádcích 1 až 3 v kódu 3.8 se nastaví zpracovávání modulem Dialog zprávám s INVITE metodou. Informativní text z řádku 9 se pak vypíše pro případ, že měření času mezi INVITE a BYE pro daný dialog bude menší, než 10 sekund. Praktická ukázka práce s modulem Dialog je v 6.4.6.[8]

3.3.7 HTTP_ASYNC_CLIENT

Modul HTTP_ASYNC_CLIENT slouží k asynchronnímu zasílání HTTP dotazů. Při asynchronním zaslání zprávy Kamailio nečeká na odpověď a začne zpracovávat další žádosti. Tato vlastnost je u návrhu firewallu žádoucí, jelikož Kamailiu na HTTP odpovědi nezáleží (alespoň ne v tomto návrhu firewallu), chce jen Ryu kontrolér informovat o bezpečnostní hrozbě. Před nahráním modulu je potřeba mít nahrán modul tm (transaction modul) a pv (pseudo-variable modul). Dále je potřeba mít v systému nainstalovány knihovny libcurl a libev.[2]

Před samotným zasláním HTTP zprávy je potřeba zadefinovat, jak bude zpráva vypadat a co se v ní bude posílat. Nastavení hlavičky a těla HTTP zprávy se uskutečňuje skrze pseudoproměnnou \$http_req():

```

1  $http_req(hdr) = $null;
2  $http_req(method) = "POST";
3  $http_req(hdr) = "Content-type: Application/json";
4  $http_req(body) = {"source_ip": 1.1.1.1/32, "reason": "flood"};

```

Kód 3.9: Nastavení těla HTTP POST zprávy

Na prvním řádku se provádí vynulování hlavičky. Na řádku 2 se nastavuje typ HTTP zprávy - POST. Na řádku 3 se definuje struktura těla zprávy, a to na JSON formát. Toto nastavení je v souladu s nastavením aplikace rest_firewall pro Ryu kontrolér. Více informací v sekci 4.2. Na čtvrtém řádku se pak definuje tělo zprávy.

Po nastavení potřebných polí hlavičky a těla HTTP zprávy je potřeba zprávu poslat. K tomu slouží funkce `http_async_query()`. Funkce používá dva parametry. První parametr je url, ve které se definuje IP adresa, port, cesta a případně další dodatečné informace. Slouží k definování hosta a přesné cesty, kam HTTP zprávu poslat. Druhým parametrem funkce je speciální metoda, ve které se zpracovává HTTP odpověď, popř. chyba/vypršení platnosti zprávy. Metodu pro HTTP odpovědi je potřeba explicitně vytvořit v Kamilio konfiguračním souboru.

```
1 http_async_query("192.168.100.20:8080/firewall/rules/all", "HTTP_REPLY");
2
3 route[HTTP_REPLY] {
4     if ($http_ok) {
5         xlog("L_INFO", "route[HTTP_REPLY]: status $http_rs\n");
6         xlog("L_INFO", "route[HTTP_REPLY]: body $http_rb\n");
7     } else {
8         xlog("L_NOTICE", "route[HTTP_REPLY]: error $http_err\n");
9     }
10 }
```

Kód 3.10: Zpracování HTTP POST zprávy a odpovědi v Kamilio konfiguračním souboru

Na prvním řádku kódu 3.10 je použita funkce `http_async_query()`. Ta posílá HTTP POST zprávu, definovanou v 3.9, a to na hosta 192.168.100.20 (IP adresa počítače, kde operuje Ryu kontrolér), port 8080 a cestu `/firewall/rules/all`. Ke zpracování odpovědi slouží metoda `HTTP_REPLY`, která je definována na řádcích 3 až 10. Podmínka s pseudoproměnnou `$http_ok` bude splněna, pokud přijde HTTP odpověď v pořádku. Poté se do terminálu vypíše status a tělo příchozí HTTP odpovědi, a to skrze pseudoproměnné `$http_rs` a `$http_rb`. Pokud se podmínka nesplní a HTTP odpověď v pořádku nepříjde, vykoná se řádek 8, tj. vypíše se chybová hláška HTTP odpovědi skrze pseudoproměnnou `$http_err`. V práci je výpis těla zprávy zakomentován, pro funkci firewallu nehraje roli. Praktická ukázka použití je v podstatě v celé praktické části diplomové práce, jelikož je zasílání HTTP zpráv využito k informování Ryu kontroléru o detekovaných hrozbách.[2][12]

3.3.8 JSONRPC-S

Tento modul poskytuje Kamilio RPC rozhraní skrze JSONRPC. Volání JSONRPC příkazů lze provádět pomocí více způsobů. K těmto způsobům patří FIFO soubor, UDP skrze různé sockety a HTTP/HTTPS. V práci je využit přístup FIFO pro lokální použití, a HTTP pro vzdálený přístup. Lokální přístup byl využíván jen pro testování během návrhu řešení. V případě použití HTTP/HTTPS je nutné nahrát modul `xHTTP` před tímto modulem, jinak není modul závislý na žádných dodatečných modulech ani knihovnách.[9]

Typ přístupu se nastavuje skrze parametr *transport*. Základní nastavení je *transport* ("6"), což značí FIFO a DATAGRAM (UDP) přístup. Pro HTTP a FIFO přístup je parametr nastaven na číslo 3.

K přístupu na JSONRPC rozhraní se používá funkce *jsonrpc_dispatch()*. Tato funkce zpracovává JSON-RPC požadavky, resp. generuje odpovědi. Na stránkách JSONRPC-S modulu je k dispozici upravená event metoda modulu xHTTP, která zpracovává správné i špatné volání JSONRPC.

```
1  modparam("jsonrpcs", "transport", 3)
2
3  event_route[xhttp:request] {
4      if(!(allow_source_address("100"))) {
5          xhttp_reply("403", "Forbidden", "text/html",
6              "<html><body>Not allowed from $si</body></html>");
7          exit;
8      }
9      if ($hu =~ "^/RPC") {
10         xlog("jsonrpc call from $si");
11         jsonrpc_dispatch();
12     } else {
13         xhttp_reply("200", "OK", "text/html",
14             "<html><body>Wrong URL $hu</body></html>");
15     }
16     return;
17 }
```

Kód 3.11: Zpracování JSONRPC volání

Na řádku 1 je nastavení parametru pro FIFO a HTTP/HTTPS přístup. Řádek 3 definuje event metodu modulu xHTTP. Tato metoda zpracovává HTTP požadavky a je volána vždy, když je zpracováván nový HTTP požadavek. Řádky 4 až 8 slouží k pokusu o zabezpečení přístupu k JSONRPC rozhraní. Originální metoda se vztahuje pouze na localhost IP (127.0.0.1). Podmínka byla tedy upravena tak, aby měl k JSONRPC rozhraní přístup jen ten, kdo má svou IP adresu ve skupině číslo 100 definované modulem Permissions. Řádek 9 až 11 provádí funkci *jsonrpc_dispatch()* v případě, že je HTTP žádost poslána na správnou URL s koncovkou /RPC. K příjmu JSONRPC příkazů je na Kamailio serveru otevřen port 8081. Pokud URL není správná, vrátí metoda zprávu 200 OK s tělem Wrong URL (špatná URL), a vypíše také URL, kterou odesílatel poslal (skrze pseudoproměnnou \$hu).[9]

Kapitola 4

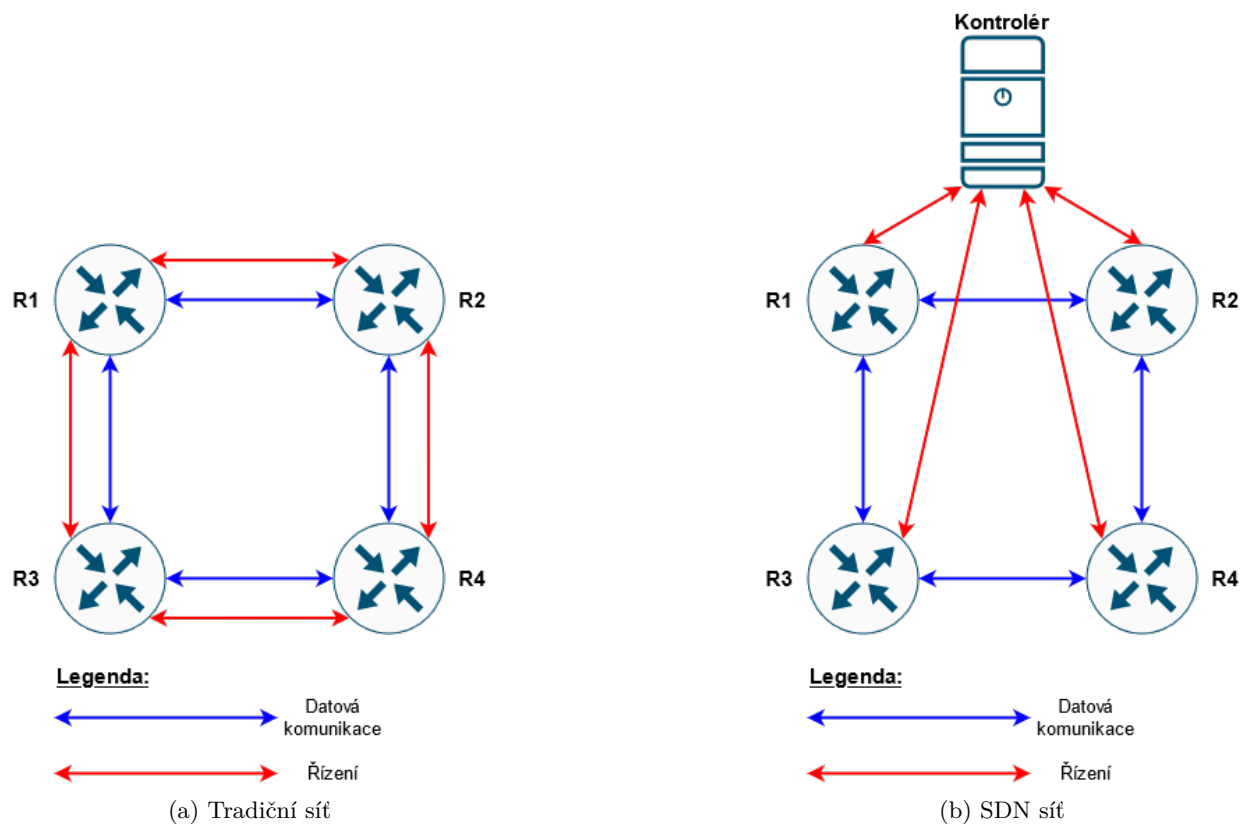
SDN

Softwarově definovaná síť (SDN) je poměrně moderní paradigma počítačových sítí, které se do povědomí veřejnosti dostalo okolo roku 2010. Úkolem SDN je umožnit centrální správu a programovatelnost sítí, resp. zvýšit jejich flexibilitu. Centrální správa síťových zařízení je uskutečněna pomocí hlavního řídicího prvku sítě - kontroléru. Kontrolér ke správě zařízení využívá příslušné síťové protokoly. Tyto protokoly vznikaly a vznikají postupně s vývojem SDN (např. protokol OpenFlow). V SDN infrastruktuře se ale také využívají protokoly z tradičních sítí (např. SNMP). Programovatelnost a flexibilitu představují SDN aplikace, které se starají o inteligenci, správu a monitoring sítí. Aplikace jsou obvykle přítomny v systému spolu s kontrolérem. Kontrolér a aplikace spolu komunikují pomocí API. Jedna z největších výhod SDN je možnost konzistentní správy sítě i v komplexním prostředí s mnoha složitými technologickými celky.[18][19]

SDN nemá jednotnou definici a každý síťový poskytovatel, výrobce či vývojář má trochu jinou představu o tom, co si pod pojmem SDN představit. Nejčastější spojitost mezi různými interpretacemi SDN je rozdělení do tří úrovní:

- **Application Plane - aplikační úroveň** se stará o definování logiky všech procesů spojených se správou sítě; je obvykle přítomna v systému, kde je umístěn samotný kontrolér; komunikace mezi aplikační (aplikací) a kontrolní (kontrolérem) úrovní probíhá skrze severní (Northbound) rozhraní - pomocí API
- **Control Plane - kontrolní úroveň** se stará o vykonávání logiky definované aplikacemi - sběr dat ze síťových zařízení, vyhodnocování dat, definování přepínacích tabulek pro spravovaná síťová zařízení apod.; vytváří můstek mezi aplikační a datovou úrovní
- **Data Plane - datová úroveň** se stará o samotné přepínání dat na základě přepínacích pravidel, které přijímá z kontrolní úrovně; jako datová úroveň je v rámci SDN paradigmatu často znázorňován přepínač, který postrádá kontrolní úroveň a je spravován kontrolérem

Na obrázku 4.1 je znázorněn rozdíl mezi tradiční a SDN sítí. Síťová zařízení tradiční sítě fungují jako samostatné celky - disponují kontrolní i datovou úrovní. Udržují si vlastní statistiky, monitoring, sousedství s vedlejšími zařízeními a zpracovávají veškerou výměnu informací se všemi prvky sítě, se kterými je to potřeba. Tyto úkony jsou souhrnně pojmenovány jako kontrolní, v obrázku jsou uvedeny jako „řízení“. Po konfiguraci zařízení sítě, navázání sousedství a např. zprovoznění dynamického směrovacího protokolu dokáže síť bez problému fungovat. Pokud je však potřeba rychlého zásahu do konfigurace, změny topologie, zjištění statistik/monitorování apod., je to vždy problém a mimo jiné je nutno zasahovat do každého ovlivněného zařízení zvlášť. Naproti tomu SDN přístup ke všem těmto kontrolním úkonům využívá kontrolér. Na síťových zařízeních je potřeba mít zprovozněnu datovou úroveň a API modul, resp. rozhraní, skrze které spolu zařízení a kontrolér budou komunikovat. Logika směrování, výměny dat, blokování provozu a statistik/monitorování se řeší centrálně. Pokud jsou v SDN architektuře chytře navrženy a zároveň propojené aplikace, dokáže se síť flexibilně adaptovat na různé změny. Mezi další přednosti SDN patří efektivní přizpůsobení přenosové kapacity linek a konzistentnost.[18][19]



Obrázek 4.1: Srovnání tradiční a SDN sítě

4.1 Kontrolér

Jak již bylo řečeno na začátku kapitoly, kontrolér představuje centrální prvek SDN sítě. Kontrolér komunikuje s připojenými síťovými zařízeními skrze jižní (Southbound) rozhraní. S aplikací komunikuje skrze severní (Northbound) rozhraní. Hlavním cílem kontroléru je sběr dat ze spravovaných zařízení, vytváření komplexních analýz nad těmito daty a na základě výsledných hodnot zpětné ovlivňování chování zařízení. Obecný kontrolér by měl mít tyto vlastnosti a funkce:

- Správa stavu sítě s možností ukládání dat do databáze a provádění úkonů nad těmito daty
- Vysoko-úrovňový datový model, který zachytává vztah mezi dostupnými prostředky, politikami a ostatními službami, které kontrolér spravuje
- Moderní Northbound API, pomocí kterého kontrolér vystavuje své služby aplikacím
- Zabezpečenou TCP relaci mezi kontrolérem a síťovými hosty
- Standardizovaný protokol pro zajištění stavu jednotlivých síťových zařízení na základě aplikace
- Mechanismus pro detekci zařízení a služeb, zjišťování topologie, výpočet nejlepších cest či dostupných síťových zdrojů

Ve většině případů není možné, aby tolik funkcí zastával pouze jeden systém (jeden kontrolér). Proto se spíše používá model více systémů, které dohromady zastávají popsané funkce.

V dnešní době je k dispozici velký počet SDN platform a kontrolérů - opensource, proprietární, placené i zdarma. V rámci práce není možné popsat všechny. V teoretické části jsou popsány tři opensource, volně dostupné kontroléry, které jsou kompatibilní s virtuálním přepínačem Open vSwitch. Jsou to OpenDaylight, ONOS a Ryu. Kontroléry byly vybrány na základě četnosti výskytu v odborných článcích a elektronických publikacích, aktivitě vývojářů a fanouškovské základně. OpenDaylight a ONOS jsou popsány obecně, Ryu je popsán podrobněji z důvodu využití tohoto kontroléru v praktické části.[18]

4.1.1 OpenDaylight

OpenDaylight je open-source projekt pod záštitou organizace The Linux Foundation. Slouží jako platforma pro SDN sítě. První zmínky o OpenDaylight spadají do poloviny roku 2013. Do dnešní doby bylo vydáno 13 verzí OpenDaylight, přičemž jsou všechny pojmenovány po chemických značkách. Poslední stabilní verze, uvolněná ke konci roku 2020, se jmenuje Aluminium (Hliník). OpenDaylight projekt používá vlastní OpenDaylight kontrolér, založený na JVM (Virtuální stroj se základem v programovacím jazyce Java). Je spustitelný na jakémkoliv operačním systému, který Javu podporuje. OpenDaylight kontrolér obecně využívá tyto čtyři nástroje:[20][21]

- **Apache Maven** - slouží ke snadnější automatizaci instalace, skriptování závislostí mezi balíčky a popisu toho, jaké balíčky se mají načítat a spouštět
- **OSGi** - tento Java framework slouží jako backend pro celý OpenDaylight projekt. Umožňuje dynamické načítání balíčků, balíčků souborů JAR a vázání balíčků dohromady - něco, co v samotné Javě není možné
- **Java rozhraní** - slouží jako vstup pro události a zpětné volání funkcí z balíčků v reakci na příchozí události
- **REST API** - slouží jako Northbound rozhraní

Komunikace s aplikacemi probíhá skrze OSGi nebo REST API. Komunikace se síťovými zařízeními zajišťují síťové protokoly. Ty jsou do OpenDaylight kontroléru přidávány pomocí pluginů. Pokud není nahrán žádný dodatečný plugin, kontrolér bude používat OpenFlow 1.0. Mezi podporované protokoly patří např. OpenFlow 1.3, BGP-LS, NETCONF, LISP, OVSDB, PCEP apod.[20][21]

4.1.2 ONOS

Open Network Operating System, zkráceně ONOS, je open-source SDN platforma, která je pod záštitou organizace The Linux Foundation a její počátek se datuje do roku 2012. Poskytuje kontrolní úroveň pro SDN síť - správu síťových zařízení a spouštění programů či modulů pro potřeby komunikačních služeb pro koncové hosty a sousední sítě. Platforma se při spojení s aplikacemi chová jako rozšiřitelný, modulární a distribuovaný SDN kontrolér, který může fungovat napříč více servery. Kontrolér pak využívá procesorový výkon a paměti všech těchto serverů, podporuje odolnost proti výpadku serveru a umožňuje výměnu hardwaru či upgrade softwaru bez dopadu na síťový provoz. Samotný projekt je aktivní, má širokou komunitu vývojářů a partnerů světové úrovně (AT&T, Telefonica, Deutsche Telekom apod.) a udržované stránky s technickými detaily, návody a příklady aplikací a programů pro určité případy použití. Verze projektu jsou pojmenovány podle různých druhů ptáků, a to v abecedním pořadí. První verze je z roku 2014 s názvem Avocet (Tenkozobec), poslední vydaná verze je z roku 2020 s názvem Toucan (Tukan).

ONOS jádro a základní služby, stejně jako aplikace, jsou psány v jazyce Java jako balíčky, které jsou načteny do kontejneru Karaf OSGi. Framework OSGi, jak už bylo popsáno v 4.1.1, je systém komponent pro Javu, který umožňuje instalovat a dynamicky spouštět moduly, a to v jediném JVM. Apache Karaf je samostatný aplikační kontejner podporující širokou škálu technologií a aplikací.

Jelikož se ONOS vyvíjel primárně pro správu velkých sítí poskytovatelů služeb, počítá se s tím, že bude distribuován mezi více serverů. Neznamená to však, že není využitelný v prostředí data-centra, kampusu či enterprise. Z důvodu kritické povahy a rozsahu sítí poskytovatelů služeb je u vývoje ONOS platformy kladen největší důraz na vysokou dostupnost, škálovatelnost a výkon. Aby

byl systém ONOS škálovatelný a odolný vůči různým typům poruch, je postaven jako fyzicky distribuovaný systém. Aby se však zachovala jednoduchost centralizované řídicí úrovně, zůstává ONOS logicky centralizovaný. Proto je ONOS postaven jako pevně spojený symetrický soubor jednotlivých instancí. Tyto instance jsou navzájem softwarově identické a každá je schopna převzít pracovní zátěž jakékoli jiné instance v případě poruchy nebo při vyvažování pracovních zátěží. ONOS používá modulární jádro, kde se jednotlivé subsystémy vždy starají jen o část síťové problematiky nebo o jednotlivé služby. Jsou společně propojeny API rozhraními. Jádro obklopují dvě logické vrstvy - Northbound rozhraní pro komunikaci s aplikacemi a Southbound rozhraní pro komunikaci se síťovými zařízeními. Aplikace mohou využívat REST API či OSGi rozhraní. Pro síťová zařízení je potřeba nahrát odpovídající subsystémy. Mezi podporované protokoly patří OpenFlow, NETCONF, OVSDB či TL1.[22][23]

Vzhledem k tomu, že jsou platformy OpenDaylight i ONOS spravovány stejnou organizací, psány ve stejném programovacím jazyce a využívají stejné nástroje i modulární architekturu, bylo by vhodné sepsat jejich hlavní rozdíly:

	OpenDaylight	ONOS
License	Eclipse Public License 1.0	Apache 2.0
Cíloví klienti	Menší projekty ve srovnání s ONOS, kampus/datacentra	Větší projekty ve srovnání s OpenDaylight, poskytovatelé služeb
Zaměření	Náhrada starších protokolů a architektur, migrace od tradičních sítí směrem k SDN	Výkon a klastrování, navýšení dostupnosti a škálovatelnosti

Tabulka 4.1: Srovnání největších rozdílů SDN platforem OpenDaylight a ONOS

4.1.3 Ryu

Ryu je open-source SDN framework vydaný pod licencí Apache 2.0. Slouží k usnadnění správy a adaptace síťového provozu. Mezi hlavní podporovatele patří japonská telekomunikační společnost NTT, která Ryu využívá v datacentrovém prostředí. Dále pak Cloud platforma OpenStack, se kterou Ryu spolupracuje skrze Quantum plugin. Ryu je napsán v programovacím jazyce Python, ve kterém se také píše jeho aplikace. Jelikož Python patří mezi snadnější, poměrně intuitivní vysokoúrovňový jazyk, je Ryu oblíbený a vhodný pro SDN začátečníky. Ryu framework má velmi dobře sepsané dokumentační stránky. Mimo tyto stránky existuje také příručka, spravována Ryu vývojovým týmem, jejíž poslední verze 4.34 je z února roku 2021.[25]

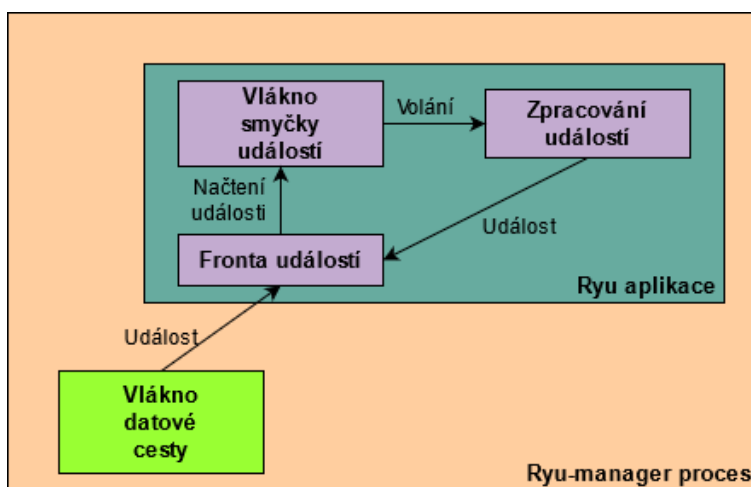
Na rozdíl od modulárních architektur OpenDaylight a ONOS kontroléru je Ryu považován za architekturu postavenou na komponentech. Největší rozdíl je v tom, že komponenta není soběstačná. Modul může a nemusí být nahrán. Pokud nahrán nebude, systém pouze přijde o funkcionalitu, kterou modul měl představovat. V případě odstranění komponenty z kódu by závislé komponenty

neměly s čím spolupracovat. Při využití Ryu co by OpenFlow kontroléru se mezi hlavní komponenty řadí:

- **ryu.base.app_manager** - načítání aplikací, poskytování kontextu aplikacím, přeposílání zpráv mezi aplikacemi
- **ryu.controller.controller** - zpracovává spojení z přepínačů, generování a směrování událostí do aplikací
- **ryu.controller.dpset** - správa přepínačů
- **ryu.controller.ofp_event** - definuje OpenFlow události
- **ryu.controller.ofp_handler** - základní operace s OpenFlow
- **OpenFlow kodéry a dekodéry** - komponenty pro OpenFlow 1.0 až 1.5

Co se týče Souhbound protokolů a správy síťových zařízení, tak má Ryu knihovny k používání OF-config, BGP, OVSDb, OpenFlow (verze 1.0 až 1.5, a také Nicira rozšíření), a k routing engine MRT.[25]

Mimo to, že je Ryu založen na komponentách, je také tzv. Event-based (řízený událostmi). Obrázek 4.2 znázorňuje pohled na programovací model používaný pro Ryu aplikace. Před spuštěním aplikace



Obrázek 4.2: Model zpracování událostí v Ryu prostředí

je potřeba v systému nainstalovat Ryu (viz 6.3). Dále je potřeba spustit Ryu-manager proces, ve kterém je definována cesta k aplikaci. Aby byl Python kód brán jako Ryu aplikace, musí dědit ze třídy *ryu.base.app_manager.RyuApp*. Události jsou instance třídy, které dědí *ryu.controller.event.EventBase*.

Komunikace mezi aplikacemi se provádí vysíláním a přijímáním událostí. Každá nahraná aplikace má svou vlastní (jednu) FIFO frontu a (jedno) vlákno. Vlákno aplikace spouští tzv. Event-loop (Vlákno smyčky událostí). Pokud je ve frontě přítomna událost, smyčka událostí jí načte a zavolá příslušnou obsluhu ve formě bloku Zpracování událostí (Event-handler). Definování konkrétní obsluhy bloku Zpracování událostí se dělá pomocí dekování třídní metody aplikace skrze `ryu.controller.handler.set_ev_cls` dekorátor. Mimo možnost napsat si vlastní aplikace na míru pro používanou síť, má Ryu několik užitečných vestavěných aplikací hned po instalaci (ve složce `ryu/app/`). Dle jejich názvu jde intuitivně poznat k čemu slouží (např. Traffic Aggregation, QoS, Traffic Monitor, Spanning Tree apod.), matoucí jsou pak koncová čísla 10, 12 a 13. Ty označují verzi protokolu OpenFlow (1.0, 1.2 a 1.3).[26]

Ryu aplikace: REST Firewall

Jedna z vestavěných aplikací Ryu frameworku je Firewall, který dokáže přijímat přesně definované příkazy pomocí REST rozhraní, a ty pak aplikovat na přepínače pomocí OpenFlow protokolu. Jedná se o jeden zdrojový kód, s délkou 1100 řádků, umístěný ve složce `/ryu/app` pod názvem `rest_firewall.py`. Po spuštění aplikace se Ryu framework chová jako OpenFlow kontrolér, ke kterému se mohou připojit OpenFlow zařízení, a to na port 6633. Následující tabulka představuje výčet příkazů, které lze na připojené přepínače skrze REST rozhraní aplikovat:[25] Součástí apli-

Název	Metoda	URL	Data
Změna stavu přepínače	PUT	<code>/firewall/module/{op}/{switch}</code>	<code>op = enable/disable</code> <code>switch = dpid/all</code>
Stav přepínače	GET	<code>/firewall/module/status</code>	
Informace o záznamu	GET	<code>/firewall/rules/{switch}/{vlan}</code>	<code>switch = dpid/all</code> <code>vlan = VLAN ID/all</code>
Smazání záznamu	DELETE	<code>/firewall/rules/{switch}/{vlan}</code>	<code>switch = dpid/all</code> <code>vlan = VLAN ID/all</code>
Stav logování	GET	<code>/firewall/log/status</code>	
Změna logování	PUT	<code>/firewall/log/{op}/{switch}</code>	<code>op = enable/disable</code> <code>switch = dpid/all</code>
Přidání záznamu	POST	<code>/firewall/rules/{switch}/{vlan}</code>	<code>switch = dpid/all</code> <code>vlan = VLAN ID/all</code>

Tabulka 4.2: Syntaxe REST příkazů Firewall aplikace

kace je spuštění WSGi web rozhraní (port 8080), skrze které se popsané operace zadávají pomocí REST API (HTTP zpráv). Pokud syntaxe příkazu vyžaduje tělo HTTP zprávy (přidávání pravidla - POST), tak musí mít JSON formát. V syntaxi lze definovat zdrojovou a cílovou MAC adresu, IP adresy verze 4 i 6, zdrojový a cílový port, transportní protokol (TCP/UDP/ICMP/ICMPv6) a typ akce - ALLOW/DENY - pro povolení/zakázání definované komunikace. V příkazech lze také rozlišovat VLAN ID. Pokud je použita síť bez VLAN technologie, tento parametr se může vyne-

chat úplně. Každý připojený OpenFlow přepínač má svůj unikátní identifikátor - DPID. HTTP příkazy se mohou soustředit na konkrétní přepínač skze tento identifikátor, nebo se může použít klíčové slovo „all“, které říká, že se bude obsah zprávy aplikovat na všechny připojené přepínače. V originálním kódu je definováno pár základních pravidel, které se aplikují na každý nově připojený přepínač - datový průchod je vypnut, logování je zapnuto, je zapnuto základní pravidlo pro ARP (L2 přepínání) a je nastavena tzv. default drop rule, což je pravidlo zahazující všechny pakety, které nesplňují žádné pravidlo v tabulce.[27]

Existují dva přístupy, jak se k firewallu stavět - buď od začátku veškerý provoz zakázat, a pak postupně jednotlivá spojení povolovat, nebo povolit vše na začátku, a pak postupně zakazovat. Rozhodnutí silně závisí na použité topologii, povaze sítě a návrhu bezpečnosti. Bezpečnější je vše zakázat a povolovat jen tzv. důvěryhodné sousedy/podsítě. Pro tento případ je nutné na firewall zadávat dvě pravidla pro jedno spojení - symetrické povolení $A \rightarrow B$, a $B \rightarrow A$, jinak obousměrná komunikace nebude fungovat. Při výskytu hrozby z již povolené komunikace stačí zablokovat pouze zdrojovou IP adresu.

Kapitola 5

Open vSwitch

Open vSwitch, někdy zkracován jako OVS či OvS, je opensource virtuální přepínač, který je vydáván pod licencí Apache 2.0. Cílem Open vSwitch projektu je vytvořit přepínací prvek produkční kvality, který podporuje standardní management rozhraní a zároveň je možné jej ovládat programově. Podstatná část kódu je napsána v jazyce C, což umožňuje virtuální přepínač instalovat na mnoho druhů platform. Open vSwitch je vhodný pro funkci virtuálního přepínače ve VM prostředí a může být spuštěn na více fyzických serverech zároveň. Ke zpracování síťového provozu využívá Kernel modul systému. Přepínač může být použit i bez tohoto modulu, avšak toto řešení je považováno za experimentální a sníží se tím výkon. Open vSwitch je veden jako základní přepínací prvek v Xen-Server 6.0 a v The Xen Cloud platformě. Je podporován ve virtualizačních platformách Xen, KVM, VirtualBox a Proxmox VE. Byl integrován do Cloud computing platform OpenStack, openQRM, OpenNebula a oVirt. Je spustitelný v operačních systémech Linux (s podporou distribucí Debian, Fedora, Ubuntu, OpenSUSE), NetBSD, FreeBSD a v prostředí Hyper-V.[29]

V práci je použit Open vSwitch přepínač verze 2.13.1, přičemž nejnovější verze současné doby je 2.15.90. Verze 2.13.x se stala v srpnu roku 2020 LTS verzí. Ke spuštění této verze je potřeba mít jádro Linuxu alespoň ve verzi 3.16. Mezi podporované funkce OvS patří:[29]

- Technologie VLAN (standard 802.1Q) - s podporou access i trunk módu
- Bonding síťových karet s možností nasazení LACP
- Technologie NetFlow, sFlow a port mirroring - pro potřeby monitorování sítě
- Konfigurace QoS, včetně možnosti využít policing
- Tunelování provozu pomocí protokolů Geneve, GRE, VxLAN, STT a LISP
- Standard 802.1ag pro management poruch spojení
- Protokol OpenFlow

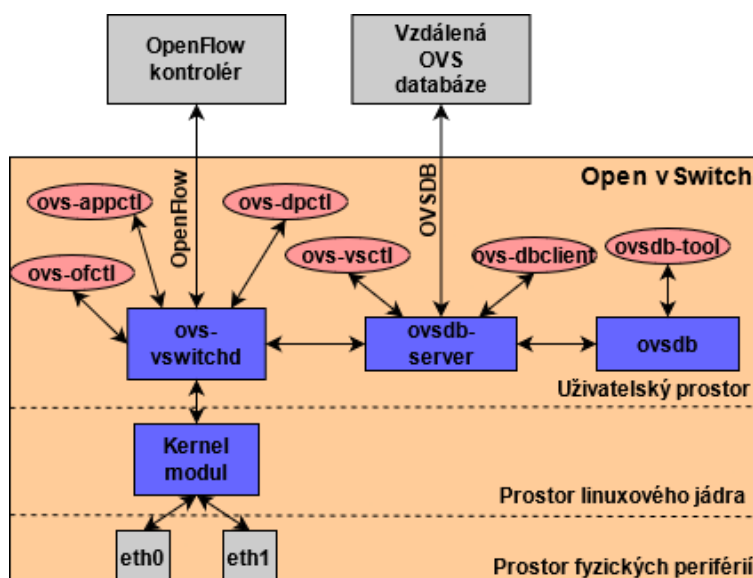
- Transakční konfigurační databáze s vazbou na jazyky C a Python
- Vysoce výkonné přepínání dat s využitím Kernel modulu

5.1 Architektura

Hlavní komponenty a nástroje Open vSwitch přepínače jsou k vidění na obrázku 5.1. Oranžovou barvou je vyobrazen Open vSwitch jako celek. Modrou barvou jsou znázorněny komponenty, červenou nástroje. Open vSwitch v systému využívá hlavně uživatelský prostor a prostor linuxového jádra (kernel prostor). Jelikož však ke své činnosti využívá také fyzická rozhraní, je zakreslen s prostorem fyzických periférií. Všechny komponenty přepínače jsou umístěny v uživatelském prostoru. V prostoru linuxového jádra je umístěn Kernel modul, se kterým OvS spolupracuje. Ústředním prvkem je `ovs-vswitchd`, což je Open vSwitch daemon. Tento daemon přebírá konfiguraci z `ovsdb` databáze skrze `ovsdb-server`. Proto jsou při spuštění Open vSwitch přepínače vždy spuštěny dva procesy - jeden pro `ovs-vswitchd`, druhý pro `ovsdb-server`. V rámci jednoho systému je povolen pouze jeden proces `ovs-vswitchd`. Ten spravuje všechny instance virtuálních přepínačů. Open vSwitch může fungovat samostatně, ale to pak musí být spravován pomocí příslušných Open vSwitch nástrojů a skriptů. Druhou možností je propojení s kontrolérem. Daemon s kontrolérem komunikuje pomocí protokolu OpenFlow. Vzdálené spojení může být realizováno nad protokoly TCP či TLS. Podporované verze OpenFlow se mohou u jednotlivých verzí OvS lišit. Od verze 2.12.x existuje podpora OpenFlow 1.0 až 1.5. Pro správu OvS verze 2.13.1, použitého v práci, by tedy mohl být použit jakýkoliv kontrolér, který nějakou z těchto verzí OpenFlow protokolu podporuje. Při startu si daemon načte konfiguraci z databáze (`ovsdb`) skrze OvS databázový server (`ovsdb-server`). Při jakékoliv následné změně databáze se daemon snaží být synchronizován, aby měl aktuální informace. Databázový server může komunikovat buď s lokální databází `ovsdb`, nebo se vzdáleným databázovým serverem skrze OVSDb protokol. Instance základního databázového serveru je v `unix:/var/run/openvswitch/db.sock`. [30][31]

5.2 Nástroje

Open vSwitch využívá okolo dvaceti nástrojů a skriptů, které se používají ke správě, testování a diagnostice systému. Na obrázku 5.1 jsou znázorněny ty nejhlavnější. Celý výpis všech nástrojů a skriptů, včetně manuálových stránek, je k nalezení na oficiálních stránkách Open vSwitch projektu. `Ovsdb-tool` slouží ke správě `ovsdb` databáze skrze příkazovou řádku. Neinteraguje ale přímo s databázovým serverem. Ke správě běžícího databázového serveru (jeho instance) pak slouží nástroj `ovs-dbclient`. Nástroj `ovs-appctl` slouží k dotazování se běžícího OvS daemona. Pomocí nástroje `ovs-dpctl` pak lze pracovat s kernel modulem a spravovat kernel datapaths. Poslední dva nástroje, `ovs-ofctl` a `ovs-vsctl`, jsou jako jediné použity v praktické části práce, budou tedy představeny blíže.



Obrázek 5.1: Hlavní komponenty a nástroje architektury Open vSwitch[30]

ovs-vsctl

Tento nástroj poskytuje vysokoúrovňové rozhraní ke konfiguraci `ovs-vswitchd` daemonu. Nástroj je připojen k procesu `ovsdb-server`, který spravuje konfigurační databázi. Pomocí tohoto připojení vyhledává a případně aplikuje změny na databázi v závislosti na zadaných příkazech. Jelikož se `ovs-vswitchd` synchronizuje s konfigurační databází, `ovs-vsctl` vždy čeká, než se po zadání příkazu daemon překonfiguruje. Mimo konfigurační příkazy může nástroj sloužit i ke kontrole, a to zejména konfigurace, vytvořených mostů a připojených rozhraní.

ovs-ofctl

`Ovs-ofctl` slouží jako příkazová řádka ke správě OpenFlow přepínače. Hlavním cílem je administrace Flow tabulky, tj. přidávání, modifikace a mazání záznamů. Nástroj také obsahuje příkazy, pomocí kterých lze diagnostikovat stav Flow tabulky a OpenFlow spojení s kontrolérem. Tento nástroj může sloužit ke správě jakéhokoliv OpenFlow přepínače, není limitován pouze na Open vSwitch.[29][31]

5.3 Flow tabulka

Flow tabulka je tabulkou záznamů, se kterými se porovnávají jednotlivé pakety příchozího síťového provozu. Tuto tabulku obsahuje každý OpenFlow přepínač. Open vSwitch podporuje až 255 tabulek. V Open vSwitch jsou tabulky procházeny sekvenčně, počínaje tabulkou číslo 0. V každé tabulce se tak může řešit jiná problematika, např. bezpečnost, následně NAT a nakonec směrování. Jednotlivé záznamy obsahují tyto části:

- **Match fields** - porovnávání pravidla proti datům z hlavičky paketu, popř. metadat z předchozích tabulek
- **Priority** - pokud by mohl paket splňovat více pravidel v jedné tabulce, vybere se ta s nejvyšší prioritou
- **Counters** - čítač dat (počet paketů/bytů), je aktualizován jen při splnění porovnání pravidla
- **Instructions** - instrukce, co s paketem provést - např. povolit, zakázat, modifikovat metadata, přeposlat do další tabulky apod.
- **Timeouts** - časovače, jsou definovány dva:
 - *idle timeout* - záznam je z flow tabulky odstraněn v případě, že po dobu tohoto časovače žádná data nesplní požadavky k použití tohoto záznamu
 - *hard timeout* - po uplynutí tohoto časovače bude záznam z flow tabulky okamžitě smazán
- **Cookie** - identifikátor záznamu zvolený kontrolérem, pro potřeby zpracování dat nepotřebný a nečitelný; slouží kontroléru ke změnám a mazání záznamů z tabulky a provádění statistik

Pokud jsou idle a hard časovače nastaveny na 0, bude záznam v tabulce na neomezenou dobu. V tabulce existuje jeden speciální záznam, tzv. table-miss flow, který obsahuje negaci všech ostatních záznamů a má prioritu 0. Pokud příchozí paket nenajde shodu v žádném záznamu, aplikuje se tento záznam a paket bude zahozen. Tento záznam lze smazat pomocí OpenFlow kontroléru, každopádně bude funkcionality zahazování paketů zachována.[32]

Kapitola 6

Praktická část

6.1 Návrh zabezpečení

Z názvu práce je patrné, že navržené řešení musí umět detekovat hrozby a útoky na aplikační vrstvě. Oním aplikačním protokolem, v rámci kterého se provádí kontrola hrozeb, je protokol SIP. Síťová entita, která detekci provádí, je SIP server Kamailio. Jak již bylo řečeno v 1, v této práci není možné popsat a umět zachytit všechny reálné SIP hrozby a útoky. Jsou proto vybrány jen některé. Výběr hrozeb je inspirován článkem z oficiálních stránek Kamailio serveru - „Přehled fragmentů konfigurace souvisejících se zabezpečením“ (zdroj: [16]). Bezpečnostní aspekty, které práce řeší, jsou tedy tyto:

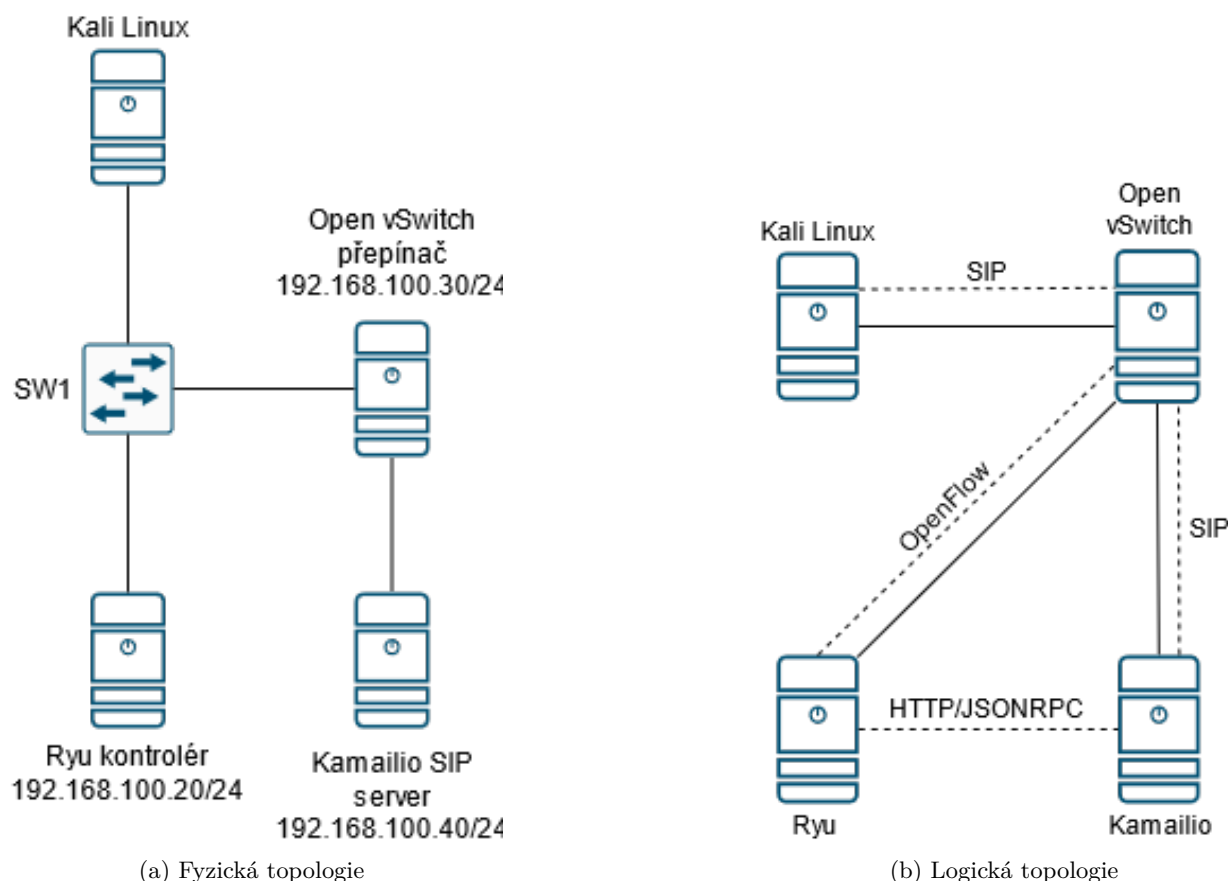
- **Útok typu Flood** - záplava SIP zpráv, nezávisle na typu SIP zprávy - a to z jednoho či více zdrojů
- **SQL injekce** - injekce vlastního příkazu do SQL databáze skrze neošetřený vstup
- **Skenování sítě** - skenování v síti, zjišťování dostupných zařízení a otevřených portů
- **VoIP spam** - časté, krátké hovory
- **SIP zpráva z nepovoleného státu** - SIP zprávy původem ze státu, odkud je komunikace nežádoucí
- **Odhalení pozměněné/poškozené SIP zprávy** - mechanismus kontroluje pouze některé prvky SIP hlavičky
- **Pokus o nepovolenou registraci účastníka**

Mimo zmíněné útoky a hrozby byla zprovozněna možnost nastavit podnikovou podsít, a k ní související povolené síť/podsítě, a to skrze Kamailio nástroj kamctl. O této implementaci pojednává sekce 6.4.10.

Kapitola 3.3 teoreticky rozebírá Kamailio moduly, které k detekci zvolených hrozeb slouží. Popis konfigurace a testování zabezpečení je v kapitole 6.4.

6.2 Topologie

Praktická část práce byla vykonávána v rámci domácí sítě s privátní adresací 192.168.100.0/24. Byly použity čtyři stěžejní entity sítě: SIP server Kamailio, SDN kontrolér Ryu, virtuální přepínač Open vSwitch a počítač s operačním systémem Kali linux, který byl využit jako útočník. Na obrázku 6.2 je znázorněna použitá topologie sítě - jak fyzický, tak logický pohled. Na obrázku 6.1a je



Obrázek 6.1: Použitá topologie při návrhu firewallu

vyobrazeno fyzické zapojení, které obsahuje čtyři fyzické počítače a jeden L2 přepínač označený jako SW1. Jelikož Open vSwitch přepínač v návrhu řešení funguje jako firewall, měl by kontrolér Ryu být připojen až za něj, aby využil blokovacího potenciálu a nebyl vystaven útočníkům. Jelikož ale domácí počítače mají pouze jednu síťovou kartu a jediný dostupný USB/RJ45 adaptér byl použit k připojení Kamailio serveru za Open vSwitch přepínač, je Ryu připojen k L2 přepínači. Ryu, Open

vSwitch i Kmailio mají statickou IP adresu. Útočník, znázorněn jako Kali Linux, žádnou konkrétní IP adresu zapsanou nemá, jelikož se v průběhu testování často mění.

Na obrázku 6.1b je znázorněn logický pohled na použitou topologii. Je nutno podotknout, že ačkoliv byl pro jednoduchost z nákresu odstraněn L2 přepínač a Ryu kontrolér je znázorněn jako přímo připojený k Open vSwitch přepínači, podle fyzické topologie tomu tak není. Plné čáry znázorňují fyzické médium, tečkované pak protokolové spojení. Mezi útočník, Open vSwitch přepínačem a Kmailio serverem se zasílají SIP zprávy, ať už za účelem hovoru či útoku. Pokud Kmailio odhalí jednu z hrozeb, pošle na Ryu kontrolér konkrétní HTTP zprávu, která má v těle definový zdroj útoku (zdrojovou IP adresu) a klíčové slovo, které označuje typ útoku. JSONRPC je v diagramu znázorněn z toho důvodu, že je použit pro účely počátečního nastavení sítě (které je popsáno v 6.4.10).

6.3 Instalace

Všechny entity sítě, kromě útočníka, byly nainstalovány na počítače s linuxovou distribucí Mint Ulyana, s linuxovým jádrem 5.4.0-58. Entity jsou vedeny jako systémové služby a jsou spravovány systémovým daemonelem systemd.

Kmailio

Kmailio je nainstalováno z repositářů. Pomocí příkazu na řádce 1 se provede synchronizace seznamu balíčků a repositářů. Na řádce 2 je samotná instalace Kmailia. Tímto příkazem se nainstaluje nejnovější verze Kmailia, která je zároveň stabilní. Pro instalaci konkrétní verze je potřeba pozměnit nastavení konfiguračního souboru nástroje APT, nebo Kmailio instalovat ze zdrojových souborů. Řádek 3 slouží pro povolení zapnutí serveru po restartu systému, řádek 4 pak ke startu serveru. V případě, že z nějakého důvodu chybí Kmailio modul, může být dohledám pomocí příkazu na řádce 5 a nainstalován pomocí příkazu na řádce 6 (zde se jedná o Geoip2 modul). Příkazy na řádcích 7 až 9 slouží k základní diagnostice v případě výskytu problému. Příkaz z řádku 7 je obecným kontrolním příkazem nástroje systemctl. Řádek 8 definuje konkrétní kontrolní příkaz, v němž parametr -ddd nastavuje úroveň kontrolních výpisů serveru (počet písmen d je přímo úměrný úrovni), -E nastavuje výpis do terminálu a -e slouží pro barevné rozlišení jednotlivých kontrolních zpráv. Parametr -c v příkaze z řádku 9 provádí kontrolu správnosti syntaxe hlavního konfiguračního souboru.

```
1 apt update
2 apt-get install kmailio
3 systemctl enable kmailio
4 systemctl start kmailio
5 apt search kmailio
```

```
6 apt install kamailio-geoip2-modules
7 systemctl status kamailio
8 kamailio -ddd -E -e
9 kamailio -c
```

Kód 6.1: Instalace Kamailio serveru

Open vSwitch

Open vSwitch přepínač je také instalován z repositářů. Na řádce 1 je provedena instalace samotného přepínače a jeho nejběžnějších nástrojů (viz 5.1). Pro potřeby dohledání dalších modulů a balíčků slouží příkaz na řádce 2, pro instalaci pak příkaz na řádce 3 (zde se jedná o instalaci balíčku pro podporu IPSec protokolu). Ihned po instalaci se spouští dva hlavní procesy - vswitchd a ovsdb-server. Pro základní kontrolu nástrojem systemctl slouží příkazy na řádcích 4 a 5.

```
1 apt-get install openvswitch-switch openvswitch-common
2 apt search openvswitch
3 apt-get install openvswitch-ipsec
4 systemctl status ovs-vswitchd
5 systemctl status ovsdb-server
```

Kód 6.2: Instalace Open vSwitch přepínače

Ryu

Jakožto framework, napsaný v jazyce Python, je Ryu stažen a instalován skrze správce balíčků jazyka Python - PIP. Na řádce 1 je stažení a instalace celého frameworku. Jelikož se Ryu spouští jako program, nemá v základu žádný proces nebo daemona, které by šlo kontrolovat skrze systemctl. Příkaz na řádce 2 vypíše verzi Ryu frameworku. Pomocí příkazu na řádce 3 se do Ryu frameworku nahrává soubor, který slouží jako SDN aplikace. Spouští se soubor s příponou .py, jelikož se aplikace píšou v jazyce Python.

```
1 pip install ryu
2 ryu-manager --version
3 ryu-manager aplikace.py
```

Kód 6.3: Instalace Ryu frameworku

Pro přidání Ryu, resp. Ryu aplikace, do daemona systému Linux (systemd), a tím vytvoření nové systémové služby, slouží kód 6.4. Pomocí řádku 1 se otevře editor souboru restfirewall.service v příslušné složce systemd. Přípona .service systému říká, že se jedná o službu, restfirewall je zvolené pojmenování dané služby. Řádky 3 až 14 obsahují tři sekce - Unit, Service a Install a slouží pro konfiguraci služby. Na řádce 4 je popis služby, řádek 6 provádí spuštění Ryu aplikace rest_firewall.py uvnitř Ryu-manager procesu. Řádky 7 a 8 slouží pro zobrazení výpisů aplikace při používání nástroje systemctl. Řádek 10 říká, že bude služba spouštěna při běžném startu systému.

```
1 nano /lib/systemd/system/restfirewall.service
2
3 [Unit]
4 Description=REST Firewall application for RYU controller
5 [Service]
6 ExecStart=/usr/local/bin/ryu-manager /root/rest_firewall.py
7 StandardOutput=syslog
8 StandardError=syslog
9 [Install]
10 WantedBy=multi-user.target
11
12 systemctl daemon-reload
13 systemctl enable restfirewall.service
14 systemctl status restfirewall.service
```

Kód 6.4: Přidání Ryu aplikace do systemd

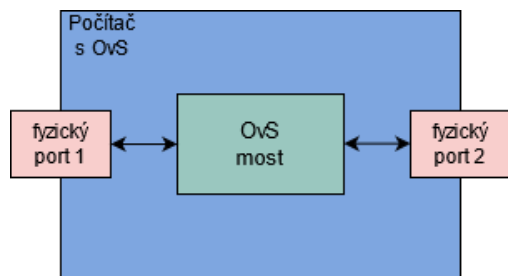
Po uložení souboru je potřeba znovu načíst daemona systemd, k tomu slouží řádek 12. Pomocí řádku 13 se povolí spuštění služby po restartu systému, pomocí příkazu na řádku 14 se pak služba zapne. Výsledek je k vidění na obrázku A.3 v příloze A.

6.4 Konfigurace a testování

V této sekci bude postupně popsána konfigurace a testování výsledného řešení aplikačního firewallu. Ke konfiguraci Kamailio serveru sloužil soubor kamailio.cfg, pro Ryu kontrolér pak soubor rest_firewall.py. Oba soubory v základu obsahují již předpřipravený kód, v práci je však popsán jen ten, který je pro návrh aplikačního firewallu klíčový. Celé konfigurační soubory jsou v dispozici v příložených souborech k diplomové práci (viz C).

6.4.1 Nastavení Open vSwitch mostu

Než se přejde ke konfiguraci zabezpečení, je potřeba nastavit OvS přepínač do stavu, kdy bude schopen přepínat pakety mezi fyzickými porty na základě pravidel, které mu předá Ryu kontrolér. K propojení dvou a víc portů, ať už fyzických nebo virtuálních, slouží tzv. most (bridge). Na obrázku 6.2a je znázorněno, jak most obecně funguje. Na obrázku 6.2b je výpis příkazu *ovs-vsctl show*, který zobrazuje konkrétní nastavení použité v této práci. Na řádcích 1 až 4 kódu 6.5 je povolení fyzických portů eno1 (směrem k útočníkovi) a enx00133b9c88dd (směrem ke Kamailiu) a vymazání jejich nastavené IP adresy. Pokud byly údaje o IP adrese přiděleny protokolem DHCP, může se také použít příkaz *dhclient -r*. Na řádku 8 je znázorněno nastavení statické IP adresy mostu. V příslušném



(a) OvS most

```

root@PC3:~# ovs-vsctl show
3ffac5c3-c0ae-44a2-b28b-3d0c5b14ef8f
Bridge bridge
  Controller "tcp:192.168.100.20:6633"
    is_connected: true
  Port bridge
    Interface bridge
      type: internal
  Port enx00133b9c88dd
    Interface enx00133b9c88dd
  Port eno1
    Interface eno1
  ovs_version: "2.13.1"
root@PC3:~#

```

(b) Ukázka příkazu ovs-vsctl show

Obrázek 6.2: Nastavení OvS mostu

zapojení je most jediným prvkem systému, který IP adresu bude mít. Na řádce 5 je vytvoření mostu s názvem *bridge*. Na řádcích 6 a 7 je přidání fyzických portů do mostu *bridge*. Na řádce 7 je nastavení spojení k OpenFlow kontroléru pro most *bridge*. Jedná se o TCP spojení na počítač s IP adresou 192.168.100.20. Spojení využívá port 6633. Příkazy řádků 10 až 12 slouží k základní diagnostice. Výpis příkazu z řádku 10 lze vidět na obrázku 6.2b. Řádek `is_connected` indikuje, že je mezi mostem a kontrolérem spojení. Příkaz z řádku 11 slouží k zobrazení základních informací o nastaveném mostu. K těmto informacím patří např. použité DPID, konkrétní možnosti nastavení mostu, možnosti a schopnosti fyzických portů apod. Poslední řádek slouží pro výpis jednotlivých záznamů z Flow tabulky mostu. Výpis tohoto příkazu lze vidět na obrázku 6.7.

```

1  ifconfig eno1 up
2  ip addr flush eno1
3  ifconfig enx00133b9c88dd up
4  ip addr flush enx00133b9c88dd
5  ovs-vsctl add-br bridge
6  ovs-vsctl add-port eno1 bridge
7  ovs-vsctl add-port enx00133b9c88dd bridge
8  ip addr add 192.168.100.30/24 dev bridge
9  ovs-vsctl set-controller bridge tcp:192.168.100.20:6633
10 ovs-vsctl show
11 ovs-ofctl show bridge
12 ovs-ofctl dump-flows bridge

```

Kód 6.5: Nastavení Open vSwitch mostu

Okamžitě po navázání spojení se mezi kontrolérem a mostem začnou posílat tzv. keepalive zprávy (obrázek v příloze A - A.2). Ty slouží k udržení spojení. Zprávy se zasílají s periodou 5 sekund, což lze v případě nutnosti přenastavit.

6.4.2 Ovládání Open vSwitch Flow tabulky skrze Ryu

Při objevení hrozby či útoku zasílá Kamailio server informace na Ryu kontrolér. Ten tyto informace zpracuje a vytvoří nový záznam ve Flow tabulce přepínače Open vSwitch. Jelikož je v práci použita vestavěná Ryu aplikace `rest_firewall`, která pro definování záznamů ve Flow tabulce přepínače používá REST rozhraní, zasílá Kamailio informace o hrozbách formou HTTP POST zpráv. Tělo HTTP zprávy má strukturu JSON formátu. Co vše lze skrze aplikaci nastavit je popsáno v 4.2. Při zaslání HTTP POST zprávy na url `http://Ryu_IP:8080/firewall/rules/all` se v aplikaci volá metoda `set_rule`. Skrze `set_rule` metodu se v Open vSwitch přepínači vytvoří nový záznam (pokud má HTTP POST zpráva správnou strukturu). Pomocí originální metody lze vytvořit dva druhy záznamu. Jeden síťový provoz povoluje, druhý blokuje. K vybrání druhu záznamu se v příkazu, který má formát JSON, využívá pole *actions*. Pro povolení provozu se používá hodnota `ALLOW`, pro blokování provozu `DENY`. Metoda byla vhodně upravena k tomu, aby se místo hodnot `ALLOW` a `DENY` mohly používat klíčová slova k určení typu hrozby. Ty jsou posílány v těle HTTP zprávy z Kamailio serveru. Dále byla do metody přidána jednoduchá logika, jak s informacemi v těle HTTP zprávy nakládat. Záznam Flow tabulky obsahuje parametr `hard timeout`, který originální metoda vždy nastavuje na 0, díky čemuž se záznam stane trvalým. Pro hodnoty `ALLOW` a `DENY` byla tato funkce ponechána, avšak pro klíčová slova hrozeb se parametr `hard timeout` dopočítává na kladnou hodnotu.

V příloze B jsou v B.1 popsány metody `which_action`, `database_update`, `action_processing` a `_to_of_flow`. První tři zmíněné jsou v aplikaci přítomny ve třídě `FirewallController`. Čtvrtá metoda se nachází ve třídě `Firewall`. Metoda `which_action`, která jako argument dostává celé tělo HTTP zprávy, se volá z metody `set_rule`. Na začátku metody `which_action` se vytváří časové razítko pro pozdější účely. Poté se vytvářejí dvě proměnné - *action* a *src_ip*, které slouží pro uložení zdrojové IP adresy a typu akce z těla HTTP zprávy. Dále má metoda za úkol zjistit, zda je proměnná *action* ekvivalentní jednomu z klíčových slov: `scanner`, `register`, `flood`, `malformed`, `geoip`, `sql` nebo `spam`. Tato klíčová slova v sobě reflektují jednotlivé útoky a hrozby ze sekce 6.1. Pokud se najde shoda, metoda provede akce z následujícího seznamu:

1. Volání třídní metody `database_update` (řádek 48 v B.1). V této metodě se pro útočníka vytvoří nový záznam v lokální Python „databázi“, pokud jeho IP nebyla po prohledání této databáze nalezena. Pokud nalezena byla, pouze se pro záznam aktualizuje časové razítko posledního útoku. Dále je zde implementována funkce mazání z databáze těch záznamů, u kterých nebylo delší dobu aktualizované časové razítko. Aby se zaručilo, že se údaje z databáze mazat budou, tak se mazání volá pokaždé, když se vytváří nový záznam v tabulce. To ale neplatí pro případ, kdy by se nový záznam vytvářel při útoku typu `flood`, jelikož by se mohlo jednat o distribuovaný `flood` útok z více IP adres, a v takovém případě je potřeba soustředit výkon

kontroléru na zpracování útoku, než na neprioritní práci s databází. V příloze A je na obrázku A.4 znázorněno vytváření/mazání záznamů v Ryu databázi.

2. Jelikož umí originální metoda pracovat pouze s hodnotami ALLOW a DENY, a v případě nalezení hrozby se bude komunikace vždy blokovat, změní se hodnota pro *actions* z klíčového slova hrozby na DENY.
3. Dle klíčového slova hrozby se v záznamu lokální databáze zvýší příslušný čítač o 1. Na základě hodnot čítačů lze vytvářet složitější logiku blokování
4. Vypočítá se hodnota proměnné „lifetime“, která slouží pro nastavení hard timeout v záznamu Flow tabulky. Výpočet provádí metoda `action_processing`, která začíná na řádce 67. V práci je výpočet pouze ilustrativní, slouží k ukázce toho, že lze aktivně vytvářet firewall záznamy jen na určitý čas. V případě útoku typu flood, nebo v případě odhalení skenování sítě, by měl být zdroj hrozby zablokován na co nejdelší dobu.

K nastavení jednotlivých parametrů záznamu slouží metoda `_to_of_flow`, která začíná na řádce 86 v B.1. Parametr hard timeout je načítán z instancí proměnné třídy Firewall. Tato proměnná je nastavována z těla metody `set_rule`, a to po výpočtu metodou `action_processing`.

V případě, že se v těle HTTP zprávy žádná klíčová slova označující hrozbu nenajdou, to metoda `which_action` pouze oznámí do konzole, nastaví hodnotu proměnné `lifetime` na 0, a vrátí nedotknuté tělo HTTP zprávy, spolu s proměnnou `lifetime`, zpět metodě `set_rule`. Na obrázku 6.3 je ukázka

```
POST /firewall/rules/all HTTP/1.1
Host: 192.168.100.20:8080
Accept: */*
Content-type: Application/json
Content-Length: 69

{"nw_src": "192.168.100.10/32", "actions": "FLOOD", "priority": "10"}HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 118
Date: Sun, 18 Apr 2021 19:07:26 GMT

[{"switch_id": "000000133b9c88dd", "command_result": [{"result": "success", "details": "Rule added. : rule_id=312"}]}
```

Obrázek 6.3: Ukázka odchycené komunikace mezi Kamailio serverem a Ryu kontrolérem

zachycení komunikace programem Wireshark. Komunikace je odchycena mezi Kamailio serverem a Ryu kontrolérem. Z těla HTTP zprávy lze vyčíst odhalení útoku typu flood, že byl útočníkem host 192.168.100.10, a nastavení priority záznamu na hodnotu 10. V rámci `rest_firewall` aplikace lze jednotlivým záznamům nastavovat prioritu, a to v rozmezí od 0 do 65533. Pomocí priorit lze upřednostňovat konkrétní OpenFlow záznamy. V rámci Open vSwitch přepínače dále existují skryté záznamy, které mají prioritu vyšší, než je nejvyšší možná hodnota pro OpenFlow záznamy. Tyto záznamy jsou však pro návrh firewallu nepodstatné. Pro zobrazení celé Flow tabulky, včetně skrytých záznamů, slouží příkaz `ovs-appctl bridge/dump-flows bridge`. V práci jsou využity tři druhy priorit.

Domácí síť je nastavena na prioritu 5. Při detekci hrozby se vytvoří záznam k blokování provozu, a to s prioritou 10. Poslední využití priority je pro spojení mezi Ryu a Kamailio serverem, a to s hodnotou 1000. Hodnoty byly zvoleny pouze orientačně.

Ve druhé části obrázku 6.3 je znázorněn text odpovědi na HTTP POST zprávu zaslanou Kamailio serverem. Odpověď informuje o úspěšném vytvoření nového záznamu č. 312 ve Flow tabulce přepínače, který má DPID 000000133b9c88dd. Zasílání této informace zpět na Kamailio se v této práci nijak nevyužívá, každopádně bylo ponecháno a může být využito v jiných pracích.

Jak bylo řečeno v sekci 2.2, při nastavování záznamů Flow tabulky využívá rest_firewall aplikace tři OpenFlow zprávy - Multipart Request, Multipart Reply a Flow Mod. Na obrázku 6.4 je k vidění první Multipart Request zpráva. Z obrázku 6.4 lze vyčíst, že je použit protokol OpenFlow verze

```

▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_MULTIPART_REQUEST (18)
  Length: 56
  Transaction ID: 251878403
  Type: OFPMP_FLOW (1)
  ▼ Flags: 0x0000
    .....0 = OFPMPF_REQ_MORE: 0x0
  Pad: 00000000
  Table ID: OFPTT_ALL (255)
  Pad: 00000000
  Out port: OFPP_ANY (4294967295)
  Out group: OFPG_ANY (4294967295)
  Pad: 00000000
  Cookie: 0x0000000000000000
  Cookie mask: 0x0000000000000000
  ▼ Match
    Type: OFPMT_OXM (1)
    Length: 4
    Pad: 00000000

```

Obrázek 6.4: OpenFlow zpráva Multipart Request

1.3, jedná se o zmiňovaný typ zprávy Multipart Request, resp. OFPMP FLOW a v těle zprávy se žádá o záznamy z Flow tabulky přepínače. Jsou použity hodnoty OFPTT ALL, OFPP ANY a OFPG ANY. Ty indikují, že má přepínač zaslat záznamy ze všech svých tabulek, všech skupin a všech portů (tedy všechny záznamy). Na obrázku 6.5 je zpráva Multipart Reply, která slouží co by odpověď zprávě Multipart Request. Zpráva je opět interpretována do OFPMP FLOW. Jak bylo řečeno v sekci 2.2.1, jelikož se posílá více záznamů z tabulky, tělo odpovědi má tvar pole, což je z obrázku patrné. Jednotlivé záznamy jsou v těle zprávy označeny jako „Flow stats“. V obrázku je otevřen záznam, který je ve Flow tabulce 0, je v ní přítomen 6291 sekund a má nastavenou prioritu 65534. Dále lze vyčíst, že skrz tento záznam prošlo 1804 paketů (108240 Bytů) a slouží k povolení průchodu protokolu ARP.

Na obrázku 6.6 je zpráva Flow Mod. Ta slouží k vytvoření, modifikaci či odstranění záznamu z Flow tabulky. V práci je Flow Mod zpráva nejčastěji využita k přidání nového záznamu. Vytvořené záznamy mají omezenou životnost skrze nastavený parametr hard timeout a mažou se samy.

```

▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_MULTIPART_REPLY (19)
  Length: 480
  Transaction ID: 251878403
  Type: OFPMP_FLOW (1)
  ▼ Flags: 0x0000
    .....0 = OFPMPF_REPLY_MORE: 0x0
  Pad: 00000000
  ▼ Flow stats
    Length: 88
    Table ID: 0
    Pad: 00
    Duration sec: 6291
    Duration nsec: 336000000
    Priority: 65534
    Idle timeout: 0
    Hard timeout: 0
    ▼ Flags: 0x0000
      .....0 = Send flow removed: False
      .....0.. = Check overlap: False
      .....0.. = Reset counts: False
      .....0... = Don't count packets: False
      .....0.... = Don't count bytes: False
    Pad: 00000000
    Cookie: 0x0000000000000000
    Packet count: 1804
    Byte count: 108240
  ▼ Match
    Type: OFPMT_OXM (1)
    Length: 10
    ▼ OXM field
      Class: OFPXMC_OPENFLOW_BASIC (0x8000)
      0000 101. = Field: OFPMT_OFB_ETH_TYPE (5)
      .....0 = Has mask: False
      Length: 2
      Value: ARP (0x0806)
      Pad: 000000000000
  ▼ Instruction
    Type: OFPIT_APPLY_ACTIONS (4)
    Length: 24
    Pad: 00000000
    ▼ Action
      Type: OFPAT_OUTPUT (0)
      Length: 16
      Port: OFPP_NORMAL (4294967290)
      Max length: 65509
      Pad: 000000000000
  ▶ Flow stats
  ▶ Flow stats
  ▶ Flow stats
  ▶ Flow stats

```

Obrázek 6.5: OpenFlow zpráva Multipart Reply

V těle zprávy lze vidět důležité údaje, jako třeba příkaz `OFPPFC ADD`, který slouží k vytváření záznamů. Dále zmíněné hodnoty časovačů, kde je pole `hard timeout` nastaveno na hodnotu 20. To značí, že se záznam po 20 sekundách smaže. Priorita je nastavena na 10. Jelikož se jedná o příkaz pro vytvoření záznamu, port a skupina se nastaví na `ANY`. Konkrétní hodnoty se nastavují pouze při mazání záznamu. V části `Match` lze vidět, že se bude přidávat záznam pro zdrojovou IP adresu (`OFPCMT OFB IPV4 SRC`) 192.168.100.10 s maskou /32.

```

1 ovs-ofctl add-flow bridge nw_src=192.168.100.10, actions=
2 ovs-ofctl add-flow bridge nw_src=192.168.100.10, actions=drop

```

Kód 6.6: Nastavení Open vSwitch záznamu pro blokování provozu

Z obrázku 6.6 nelze vyčíst, jaká akce se pro nový záznam bude používat. OpenFlow protokol totiž nemá definovanou akci pro blokování provozu. Pro vytvoření záznamu, který bude provoz zahazovat, se vytváří záznam bez akce. Situaci ilustruje kód 6.6, kde jsou si výsledky příkazů ekvivalentní. Pokud se skrze OpenFlow přidává nový záznam, který má síťovou komunikaci povolovat, musí mít definovanou akci, např. na `NORMAL`. Pokud žádná akce není nastavena, provoz bude zahazován. Na obrázku 6.7 lze vidět nastavené blokové záznamy Flow tabulky č. 0, které mají nastavený

```

▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_FLOW_MOD (14)
  Length: 72
  Transaction ID: 251878404
  Cookie: 0x0000000000000137
  Cookie mask: 0x0000000000000000
  Table ID: 0
  Command: OFPFC_ADD (0)
  Idle timeout: 0
  Hard timeout: 20
  Priority: 10
  Buffer ID: OFP_NO_BUFFER (4294967295)
  Out port: OFPP_ANY (4294967295)
  Out group: OFPG_ANY (4294967295)
▼ Flags: 0x0000
  .... = Send flow removed: False
  .... = Check overlap: False
  .... = Reset counts: False
  .... = Don't count packets: False
  .... = Don't count bytes: False
Pad: 0000
▼ Match
  Type: OFPMT_OXM (1)
  Length: 22
  ▼ OXM field
    Class: OFPXM_OPENFLOW_BASIC (0x8000)
    0000 101. = Field: OFPXM_OF_ETH_TYPE (5)
    .... = Has mask: False
    Length: 2
    Value: IPv4 (0x0800)
  ▼ OXM field
    Class: OFPXM_OPENFLOW_BASIC (0x8000)
    0001 011. = Field: OFPXM_OF_IPV4_SRC (11)
    .... = Has mask: True
    Length: 8
    Value: 192.168.100.10
    Mask: 255.255.255.255
    Pad: 0000

```

Obrázek 6.6: OpenFlow zpráva Flow Mod

```

cookie=0x116, duration=0.458s, table=0, n_packets=514, n_bytes=573110, hard_timeout=22, priority=10,ip,nw_src=192.168.100.170 actions=drop
cookie=0x117, duration=0.401s, table=0, n_packets=521, n_bytes=580915, hard_timeout=22, priority=10,ip,nw_src=192.168.100.171 actions=drop
cookie=0x118, duration=0.337s, table=0, n_packets=553, n_bytes=616595, hard_timeout=22, priority=10,ip,nw_src=192.168.100.172 actions=drop
cookie=0x119, duration=0.259s, table=0, n_packets=523, n_bytes=583145, hard_timeout=22, priority=10,ip,nw_src=192.168.100.173 actions=drop
cookie=0x11a, duration=0.228s, table=0, n_packets=559, n_bytes=623285, hard_timeout=22, priority=10,ip,nw_src=192.168.100.174 actions=drop
cookie=0x11b, duration=0.154s, table=0, n_packets=510, n_bytes=568650, hard_timeout=22, priority=10,ip,nw_src=192.168.100.175 actions=drop
cookie=0x11c, duration=0.088s, table=0, n_packets=533, n_bytes=594295, hard_timeout=22, priority=10,ip,nw_src=192.168.100.176 actions=drop
cookie=0x11d, duration=0.040s, table=0, n_packets=382, n_bytes=425930, hard_timeout=22, priority=10,ip,nw_src=192.168.100.177 actions=drop
cookie=0x4, duration=525.258s, table=0, n_packets=34532, n_bytes=17200673, priority=5,ip,nw_src=192.168.100.0/24,nw_dst=192.168.100.0/24 actions=NORMAL
cookie=0x0, duration=526.291s, table=0, n_packets=193, n_bytes=17048, priority=0 actions=drop
root@PC3:~# ovs-ofctl dump-flows bridge
cookie=0x0, duration=850.683s, table=0, n_packets=1563, n_bytes=93780, priority=65534,arp actions=NORMAL
cookie=0x1, duration=850.683s, table=0, n_packets=392, n_bytes=95568, priority=1000,ip,nw_src=192.168.100.20,nw_dst=192.168.100.40 actions=NORMAL
cookie=0x2, duration=850.683s, table=0, n_packets=619, n_bytes=102381, priority=1000,ip,nw_src=192.168.100.40,nw_dst=192.168.100.20 actions=NORMAL
cookie=0x4, duration=849.650s, table=0, n_packets=40461, n_bytes=20028060, priority=5,ip,nw_src=192.168.100.0/24,nw_dst=192.168.100.0/24 actions=NORMAL
cookie=0x0, duration=850.683s, table=0, n_packets=290, n_bytes=25150, priority=0 actions=drop
root@PC3:~#

```

Obrázek 6.7: Ukázka nastavení hard timeout ve Flow tabulce Open vSwitch přepínače

hard timeout na nenulovou hodnotu. Jakmile hodnota pole duration překročí hodnotu v poli hard timeout, záznam je z Flow tabulky automaticky smazán, což dokazuje zadání příkazu *ovs-ofctl dump-flows bridge* po vypršení hard timeout časovače.

Výpočet hard timeout je pro všechny útoky stejný. V rámci útoků se mění jen IP adresy, a to minimálně. Proto je dále sekcích spíše probíráán pohled na použité testovací nástroje, zachytávání útoků/hrozeb na Kamailio serveru a výpisy tohoto serveru. Hlavní konfigurační části Ryu a Open vSwitch byly popsány v této a minulé sekci. Pro celistvý pohled je v příložených souborech k dispozici konfigurační soubor Kamailia, Ryu, a odchycené komunikace programem Wireshark při jednotlivých útocích a nastaveních.

6.4.3 Útok typu Flood

Každý server, který poskytuje nějakou službu, může obsluhovat konečný počet požadavků. To indikuje, že při nadměrném zatížení server nemůže zpracovávat nově příchozí požadavky. V takovém případě se požadavky začnou vkládat do fronty, nebo rovnou zahazovat. Nabízená služba je tak nedostupná, což je cílem útoku typu flood - odepřít službu pomocí velkého množství zpráv, které nenesou žádnou informaci. V práci je nabízenou službou zpracování SIP požadavků pomocí SIP serveru Kamailio. K útoku je použit nástroj `inviteflood`, který je penetračním nástrojem linuxové distribuce Kali. Jak je z názvu nástroje patrné, cíl útoku je zasažen velkým množstvím INVITE SIP zpráv. Nastavená logika Kamailia není na typ SIP zprávy orientována. Pomocí nastavení limitace počtu příchozích zpráv za časovou jednotku se zablokuje jakýkoliv zdroj zpráv, který tento limit překročí. K odhalení útoku je využit modul PIKE (viz 3.3.1). K urychlení zahazování SIP žádostí, které jsou na server zaslány během zpracování informací o útočnickovi, slouží hash tabulka modulu HTable (viz 3.3.4).

```
1  #!/bin/bash
2  for host in {100..200}
3  do
4  ip addr flush eth0
5  ip addr add 192.168.100.$host/24 dev eth0
6  inviteflood eth0 2001 192.168.100.40 192.168.100.40 200
7  echo "$host"
8  done
```

Kód 6.7: Útok typu flood pomocí nástroje `inviteflood`

Pro účely testování firewallu byl vytvořen krátký BASH skript 6.7, který dokáže sekvenčně útočit z více IP adres. Řádky 2 až 8 označují cyklus `for`, který se provede pro čísla v intervalu 100 až 200. Útok je prováděn z fyzického portu `eth0`, který je na řádce 4 vždy vynulován. Pomocí řádku 5 se na tento port přiřadí IP adresa a maska, přičemž `host` část IP adresy je přidělena na základě konkrétního cyklu `for` smyčky. Při nastavení cyklu je potřeba dávat pozor, aby se do výběru IP adres nedostaly adresy používaných serverů. Na řádce 6 je prováděn samotný útok pomocí nástroje `inviteflood`. Útok je veden z fyzického portu `eth0`, na uživatelský účet číslo 2001, doménu 192.168.100.40, IP adresu 192.168.100.40. V rámci jednoho útoku je posláno 200 INVITE zpráv. Řádek 7 slouží ke kontrolnímu výpisu právě prováděného cyklu do terminálu. Pokud by byl skript využit mimo L2 segment domácí sítě, bylo by potřeba přidat směrování.

```
1  if($sht(ipban=>$si) != $null) {
2    #xlog("blocked..");
```

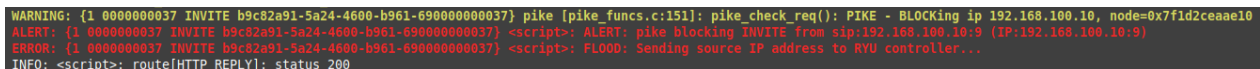
```

3   exit;
4 };
5
6  if (!pike_check_req()) {
7      if ($rc == -2) {
8          $sht(ipban=>$si) = 1;
9          $http_req(hdr) = $null;
10         $http_req(method) = "POST";
11         $http_req(hdr) = "Content-type: Application/json";
12         $http_req(body) = "{\"nw_src\": \"\"+$si+\"/32\", \"actions\": \"FLOOD\",
                             \"priority\": \"10\"}";
13         xlog("FLOOD: Sending source IP address to RYU controller...");
14         http_async_query("192.168.100.20:8080/firewall/rules/all", "HTTP_REPLY
                             ");
15         exit;
16     }
17 };

```

Kód 6.8: Zachycení útoku typu flood a zaslání informace na Ryu kontrolér

Kód 6.8 znázorňuje Kamailio kód pro zachycení útoku typu flood (řádek 6), zaslání informace o útoku na Ryu kontrolér (řádky 7 až 17) a zahazování následujících SIP zpráv ze zdrojové IP adresy, ze které byl útok poslán (řádky 1 až 4). Na Ryu se zasílá zdrojová IP adresa, klíčové slovo FLOOD a priorita 10. Jednotlivé funkční části kódu jsou více rozepsány v sekcích 3.3.1, 3.3.4 a 3.3.7. V příloze A je na obrázku A.4a znázorněno chování Ryu kontroléru při vícenásobném flood útoku. Na obrázku



```

WARNING: {1 0000000037 INVITE b9c82a91-5a24-4600-b961-690000000037} pike [pike_funcs.c:151]: pike_check_req(): PIKE - BLOCKing ip 192.168.100.10, node=0x7f1d2ceaae10
ALERT: {1 0000000037 INVITE b9c82a91-5a24-4600-b961-690000000037} <script>: ALERT: pike blocking INVITE from sip:192.168.100.10:9 (IP:192.168.100.10:9)
ERROR: {1 0000000037 INVITE b9c82a91-5a24-4600-b961-690000000037} <script>: FLOOD: Sending source IP address to RYU controller...
INFO: <script>: route[HTTP_REPLY]: status 200

```

Obrázek 6.8: Ukázka výpisů Kamailio serveru při útoku typu flood

6.8 jsou znázorněny výpisy Kamailio serveru při útoku typu flood - alarm modulu PIKE, informaci o zaslání informace o útoku na Ryu kontrolér a příjem pozitivní 200 HTTP zprávy. Na obrázku 6.7 lze vidět Flow tabulka Open vSwitch přepínače při vícenásobném použití flood útoku. Během řešení práce se vyskytl problém při zpracování velkého množství HTTP zpráv, které Kamailio server dokáže během vícenásobného útoku typu flood generovat. Problém se projevoval nastavením špatných hodnot parametru hard timeout. K odstranění problému bylo potřeba nepoužívat proměnnou *lifetime* třídy Firewall jako třídní, ale jako instanční.

V příložených souborech k této práci je odchycená komunikace programem Wireshark při útoku typu flood. Popis struktury přiloženého souboru je v příloze C.

6.4.4 Útok typu SQL injekce

Při útoku SQL injekcí se útočník snaží využít neošetřeného vstupu do SQL databáze a vložit do databáze vlastní SQL příkaz. Účastníci VoIP komunikací někde musí mít ukládány své uživatelské účty. K tomu se v drtivé většině případů používá databáze. Kontrola na SQL injektáž se v práci provádí nad uživatelským jménem SIP hlavičky. K tomuto poli lze přistupovat skrze pseudoproměnnou \$fU. Pseudoproměnnou \$au pak v případě SIP pole autorizace.

```
1 route[SQL_check] {
2   if(($fU =~ "(\\=)|(\\-\\-)|(')|(#)|(\\%27)|(\\%24)") or ($au =~ "(\\=)|(\\-\\-)|(')
      |(\\#)|(\\%27)|(\\%24)" and $au != $null)) {
3     xlog("Trying to make a SQL injection with $fU from $si:$sp\n");
4     $http_req(hdr) = $null;
5     $http_req(method) = "POST";
6     $http_req(hdr) = "Content-type: Application/json";
7     $http_req(body) = "{\"nw_src\": \"\"+$si+\"/32\", \"actions\": \"SQL\", \"
      priority\": \"10\"}";
8     http_async_query("192.168.100.20:8080/firewall/rules/all", "HTTP_REPLY");
9     sl_reply("404", "Invalid authentication");
10    exit;
11  }
12 }
```

Kód 6.9: Zachycení útoku typu SQL injekce a zaslání informace na Ryu kontrolér

V kódu 6.4.4 je definována Kamailio metoda SQL_check, která slouží k zachytávání pokusů o SQL injekci a zaslání informace na Ryu kontrolér. Jelikož se v práci předpokládá, že bude uživatelské jméno putovat do SQL databáze jen při registraci, je tato metoda volána pouze v Register metodě.

Pro testování byl použit softphone Twinkle. Na obrázku 6.9 je znázorněno nastavení příkladu správných a špatných přihlašovacích údajů. Za správné se považuje uživatelské jméno 60, což odkazuje na účet 60, který na Kamailio serveru vytvořen je. Za špatné údaje se považuje přihlašování s uživatelským jménem =. Na obrázku 6.10 je ukázka výpisů Kamailio serveru v reakci na přihlašování pod správným a špatným účtem. Jelikož byl v pseudoproměnné \$fU nalezen znak =, informuje se Ryu kontrolér a účastníkovi se zašle negativní odpověď 404 - Invalid authentication, což lze vidět na obrázku 6.11. Ekvivalentem kontroly polí fU a au je funkce secf_check_sql_hdr, resp. secf_check_sql_all Kamailio modulu Secfilter. V případě těchto funkcí jsou nepovolené znaky uloženy v tabulce databáze a při spuštění Kamailio serveru jsou nahrány do mezipaměti.

User

SIP account

Your name:

User name*:

Domain*:

Organization:

SIP authentication

Realm:

Authentication name:

Password:

User

SIP account

Your name:

User name*:

Domain*:

Organization:

SIP authentication

Realm:

Authentication name:

Password:

(a) Správné přihlašovací údaje k SIP účtu
(b) Špatné přihlašovací údaje k SIP účtu

Obrázek 6.9: Přihlašovací údaje pro testování útoku typu SQL injekce

```
ERROR: (1 902 REGISTER qidjluzagtiexbb@PC2) <script>: Register from (IP:192.168.100.20:5060).
ERROR: (1 903 REGISTER qidjluzagtiexbb@PC2) <script>: Trying to make a SQL injection with = from 192.168.100.20:5060
INFO: <script>: route[HTTP_REPLY]: status 200
```

Obrázek 6.10: Ukázka výpisů Kamailio serveru při útoku typu SQL injekce

```
Sun 22:43:25
60, registration succeeded (expires =
3600 seconds)

Sun 22:44:18
60, registration failed: 404 Invalid
authentication
```

Obrázek 6.11: Ukázka odpovědí na přihlašování pod správným a špatným uživatelským jménem

V přiložených souborech k této práci je odchycená komunikace programem Wireshark při útoku typu SQL injekce. Popis struktury přiloženého souboru je v příloze C.

6.4.5 Skenování sítě

Skenování je prvotním krokem většiny síťových útoků, jelikož se útočník potřebuje co nejlépe obeznámit se svým cílem. Hlavním účelem skenování je zjistit o síti co nejvíce informací, jako např. dostupná zařízení, používané platformy, verze operačních systémů, otevřené porty, nabízené služby, adresní plán apod. Práce se zabývá skenováním VoIP služeb. V práci je využit princip porovnávání pole User-Agent SIP hlavičky s názvy dobře známých skenovacích nástrojů. Předpoklady pro správné zachycení tedy jsou, aby sken zasáhl port, na kterém Kamailio poslouchá a aby byl v poli UA nastaven název jednoho z dobře známých skenovacích nástrojů. Pro účely testování byl nainstalován nástroj SipVicious. V rámci tohoto nástroje byl použit příkaz k mapování sítě - svmap.

```
1 if ($ua =~ "friendly|scanner|sipcli|sipvicious|VaxSIPUserAgent") {
2   xlog("Scanner detected (from $si:$sp), sending alarm to Ryu controller.");
```



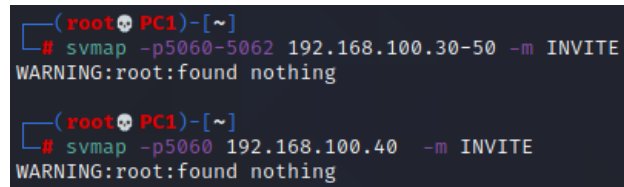
```

3  $http_req(hdr) = $null;
4  $http_req(method) = "POST";
5  $http_req(hdr) = "Content-type: Application/json";
6  $http_req(body) = "{\"nw_src\": \"\"+$si+\"/32\", \"actions\": \"SCANNER\",
    \"priority\": \"10\"}";
7  http_async_query("192.168.100.20:8080/firewall/rules/all", "HTTP_REPLY");
8  exit;
9  }

```

Kód 6.10: Zachycení skenování sítě a zaslání informace na Ryu kontrolér

V kódu 6.10 je na prvním řádku podmínka a testování pseudoproměnné \$ua (User Agent) na dobře známé názvy skenovacích nástrojů. Zbytek kódu už je jen informativní výpis o útočnickovi do terminálu a zaslání informace na Ryu kontrolér. Při detekci skenování je důležité ukončit zpracování SIP zprávy bez zasílání odpovědi zdroji této zprávy. Na obrázku 6.12 je ukázka použití skenování



Obrázek 6.12: Ukázka skenování sítě pomocí nástroje SipVicious

nástrojem SipVicious. Na IP adrese 192.168.100.40 a portu 5060 poslouchá Kamailio. Výpis nástroje ukazuje, že nebyly nalezeny žádné výsledky, což je z pohledu SIP serveru žádoucí. Na obrázku 6.13

```

ERROR: [1 1 INVITE 144598894253622238095653] <script>: Scanner detected (from 192.168.100.10:5060), sending alarm to Ryu controller.
INFO: <script>: route[HTTP_REPLY]: status 200

```

Obrázek 6.13: Ukázka výpisů Kamailio serveru při skenování sítě

je zobrazen výpis Kamailio serveru do terminálu při odhalení skenování sítě. Informace je poslána na Ryu kontrolér.

Ochrana proti skenování na základě kontroly UA je poměrně snadno porazitelná. Pokud útočník nastaví pole UA na cokoliv jiného mimo dobře známé názvy skenovacích nástrojů, kontrolou projde. Podle článku [28], kde autoři vyhodnocují data sbíraná po dobu 358 dní z celkově 50 honeypot systémů, byla nejčastěji zasílaná SIP žádost REGISTER. Povolení REGISTER zpráv na základě skupin podle modulu Permissions, jak je popsáno v 6.4.9, by tak částečně zdokonalilo ochranu proti skenování. To ale neřeší ostatní typy žádostí a možnost, že by sken prováděl někdo z povolené skupiny. V článku je dále zmíněna vlastnost automatizovaných skenovacích nástrojů měnit pole UA při každé nové zprávě. Pro komplexnější ochranu by tak bylo vhodné vytvořit mechanismus, který

by dokázal udržovat statistiku o typech a počtu přijatých SIP zpráv, nad kterou by se prováděla hlubší analýza. Pomocí této analýzy by pak za zdroj skenování mohl být označen zdroj SIP zpráv, které mají více společných vlastností. Mezi tyto vlastnosti může patřit pravidelný interval zasílání, proměnlivý obsah pole UA, časté negativní odpovědi a snaha komunikovat se všemi hosty v rámci podsítě.

V příložených souborech k této práci je odchycená komunikace programem Wireshark při odhalení skenování sítě. Popis struktury příloženého souboru je v příloze C.

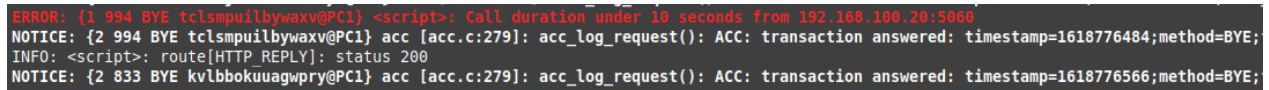
6.4.6 VoIP spam

Spam ve VoIP, označován jako SPIT, je souhrnné označení pro nevyžádané, často automaticky generované telefonní hovory, které jsou prováděny skrze Internetovou telefonii. Cílem těchto hovorů může být nabízení služeb nebo produktů, obtěžování, či pokus o různé druhy podvodů a získávání informací. V práci je využit poměrně triviální způsob odhalování spamu. SIP server Kamailio měří délku hovoru pomocí modulu Dialog (který byl popsán v 3.3.6). Pokud je délka hovoru kratší, než 10 sekund, zašle se informace na Ryu kontrolér. Aby se předešlo blokování lidí, kteří např. jen špatně zadali číslo a během prvních deseti sekund se hovor položil, je v Ryu nastavena logika blokování jen těch, u kterých se v posledních 3 minutách vyskytla aktualizace jejich záznamu v databázi. Pokud tedy někdo během tří minut provede dva a více hovorů, které mají délku trvání pod 10 sekund, bude považován za hrozbu.

```
1  if (is_method("INVITE")) {
2      dlg_manage();
3  }
4  /* code */
5  if (is_method("BYE")) {
6      if ($DLG_lifetime < 10) {
7          xlog("Call duration under 10 seconds from $si:$sp\n");
8          $http_req(hdr) = $null;
9          $http_req(method) = "POST";
10         $http_req(hdr) = "Content-type: Application/json";
11         $http_req(body) = "{\"nw_src\": \"\"+$si+\"/32\", \"actions\": \"SPAM\",
12                             \"priority\": \"10\"}";
13         http_async_query("192.168.100.20:8080/firewall/rules/all", "HTTP_REPLY")
14         ;
15     }
16 }
```

Kód 6.11: Zachycení krátkého hovoru a zaslání informace na Ryu kontrolér

Nejdůležitějšími prvky v 6.11 jsou funkce `dlg_manage` a pseudoproměnná `$DLG_lifetime`. Jejich teoretický rozbor je v sekci 3.3.6. Řádky 6 až 12 slouží k zaslání informace na Ryu kontrolér v případě, že je doba mezi INVITE a BYE zprávou menší než 10 sekund. Na řádce 40 v B.1 je pak podmínka, skrze kterou se zvýší počítadlo pro spam v rámci záznamu v databázi jen tehdy, pokud byl záznam v posledních 3 minutách aktualizován. Na prvních třech řádcích obrázku 6.14 je vidět,



```
ERROR: {1 994 BYE tclsmptulbywaxv@PC1} <script>: Call duration under 10 seconds from 192.168.100.20:5060
NOTICE: {2 994 BYE tclsmptulbywaxv@PC1} acc [acc.c:279]: acc_log_request(): ACC: transaction answered: timestamp=1618776484;method=BYE;
INFO: <script>: route[HTTP_REPLY]: status 200
NOTICE: {2 833 BYE kvlbokuuagwpry@PC1} acc [acc.c:279]: acc_log_request(): ACC: transaction answered: timestamp=1618776566;method=BYE;
```

Obrázek 6.14: Ukázka výpisů Kamailio serveru při krátkém hovoru

co se stane v případě krátkého hovoru. Podle řádku 7 v 6.11 se vypíše hláška a zašlou se informace na Ryu kontrolér. Na třetím řádku obrázku 6.14 je vidět odpověď status Ryu odpovědi. Druhý a čtvrtý řádek jsou interní hlášky modulu Dialog. V případě volání nad 10 sekund se vypíše pouze informace o metodě BYE, tak jak je vidět na řádce 4.

V příložených souborech k této práci je odchycená komunikace programem Wireshark. Při nastavení filtrů na protokol SIP a HTTP je z časů odeslání zpráv patrné, že pro krátký hovor se informace na Ryu pošlou a při dostatečně dlouhém hovoru ne. Popis struktury příloženého souboru je v příloze C.

6.4.7 SIP zpráva z nepovoleného státu

K blokaci hovorů ze zemí, ze kterých jsou hovory nežádoucí, slouží v Kamailio serveru modul Geoip2. Ten byl teoreticky probrán v 3.3.5. Po nahrání databáze, která mapuje veřejné IP adresy na geografické regiony, lze porovnávat zdrojové IP adresy hovorů s touto databází. Databáze má k daným IP adresám geografické informace, čehož je využito přímo v konfiguračním souboru Kamailia.

```
1 if (geoip2_match("$si", "src")) {
2     if($gip2(src=>cc) =~ "NG|IQ|IR|PK|ZW"){
3         xlog("IP: $si");
4         xlog("CC: $gip2(src=>cc)");
5         xlog("City: $gip2(src=>city)");
6         xlog("Zip Code: $gip2(src=>zip)");
7         $http_req(hdr) = $null;
8         $http_req(method) = "POST";
9         $http_req(hdr) = "Content-type: Application/json";
10        $http_req(body) = "{\"nw_src\": \"\"+$si+\"/32\", \"actions\": \"GEOIP\",
11                           \"priority\": \"10\"}";
12        xlog("GEOIP: Sending source IP address to RYU controller...");
```

```

12     http_async_query("192.168.100.20:8080/firewall/rules/all", "HTTP_REPLY")
13     ;
14     exit;
15 }

```

Kód 6.12: Zachycení SIP zprávy z nežádoucího státu a zaslání informace na Ryu kontrolér

Na řádku 1 kódu 6.12 je porovnání zdrojové IP adresy s geografickou databází IP adres. Výpis do terminálu a zaslání informací na Ryu kontrolér se provede tehdy, pokud se potvrdí podmínka na řádku 2. Ta bude potvrzena, pokud je zdrojová IP adresa z Nigérie, Iráku, Íránu, Pákistánu nebo Zimbabwe.

Pro otestování řešení byla vybrána pákistánská IP adresa 124.109.38.218, která byla zjištěna ze stránek pro trasování IP adres na geografické oblasti - tracemyip.org. Jelikož má domácí podsít adresaci 192.168.100.0/24, a nemám přístup na domácí router, tak byla využita možnost vytvořit virtuální IP adresu na počítači, kde je umístěn Kamailio server, a zaslat SIP zprávu v rámci jednoho systému.

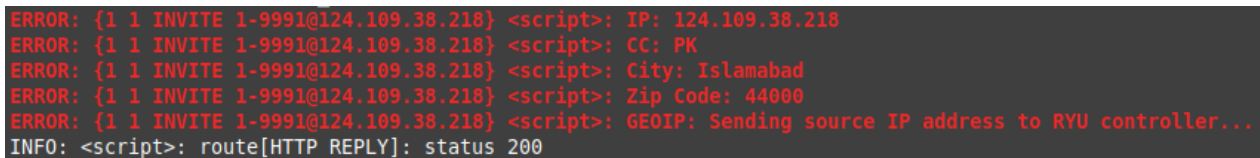
```

1 ifconfig enp3s0:0 124.109.38.218
2 sipp 192.168.100.40 -i 124.109.38.218 -r 1

```

Kód 6.13: Nastavení pákistánské IP adresy a zaslání SIP zprávy

Na řádku 1 v 6.13 je nastavení virtuální IP adresy v prostředí Linux. Na řádku 2 je pak zaslání SIP zprávy na Kamailio server se zdrojovou IP z virtuálního rozhraní pomocí nástroje SIPP. Na obrázku



```

ERROR: {1 1 INVITE 1-9991@124.109.38.218} <script>: IP: 124.109.38.218
ERROR: {1 1 INVITE 1-9991@124.109.38.218} <script>: CC: PK
ERROR: {1 1 INVITE 1-9991@124.109.38.218} <script>: City: Islamabad
ERROR: {1 1 INVITE 1-9991@124.109.38.218} <script>: Zip Code: 44000
ERROR: {1 1 INVITE 1-9991@124.109.38.218} <script>: GEOIP: Sending source IP address to RYU controller...
INFO: <script>: route[HTTP_REPLY]: status 200

```

Obrázek 6.15: Ukázka výpisů Kamailio serveru během SIP komunikace z nechtěné země

6.15 lze vidět výpis Kamailio serveru do terminálu při odhalení, že je SIP zpráva z pákistánské IP adresy. Zasílání těchto geografických informací na Ryu kontrolér by mohlo sloužit k vytváření hlubších statistik o útocích. V příložených souborech k této práci je odchycená komunikace programem Wireshark při použití pákistánské IP adresy. Přítomny jsou dva soubory, jelikož HTTP zpráva se posílá sítí na Ryu kontrolér z fyzického rozhraní, kdežto SIP zpráva z virtuální IP se posílá v rámci localhost rozhraní. Popis struktury příloženého souboru je v příloze C.

6.4.8 Odhalení pozměněné/poškozené SIP zprávy

Příchod poškozených nebo pozměněných SIP zpráv může způsobit nepředvídatelné chování systému. Pozměněné SIP zprávy mohou být také indikátorem útoku. Proto by bylo dobré umět takové zprávy odhalit. K odhalení poškozené/pozměněné SIP zprávy je využit Kamailio modul Sanity, který je popsán v 3.3.3. Je využito výchozí nastavení modulu. Konfigurace vypadá následovně:

```
1 if (!sanity_check()) {
2     xlog("Malformed SIP request from $si:$sp\n");
3     $http_req(hdr) = $null;
4     $http_req(method) = "POST";
5     $http_req(hdr) = "Content-type: Application/json";
6     $http_req(body) = "{\"nw_src\": \"\"+$si+\"/32\", \"actions\": \"MALFORMED\",
7         \"priority\": \"10\"}";
8     http_async_query("192.168.100.20:8080/firewall/rules/all", "HTTP_REPLY");
9     exit;
10 }
```

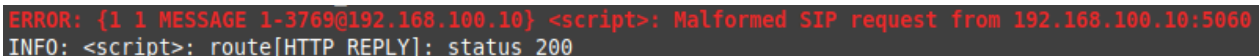
Kód 6.14: Zachycení poškozené/pozměněné SIP zprávy a zaslání informace na Ryu kontrolér

Na řádce 1 v kódu 6.14 je funkce kontroly SIP zpráv modulu Sanity. Pokud kontrola neprojde, spustí se tělo podmínky, což představuje výpis hlášky do terminálu a zaslání informací na Ryu kontrolér. Nastavení parametru modulu Sanity zůstalo základní, typy kontroly funkce sanity_check jsou popsány v 3.3.3. Pro testovací účely byl využit nástroj SIPp. Nástroj SIPp v sobě skrývá vestavěné situace pro UAC i UAS. V rámci testování byla využita šablona sipp_uac_bad_message.xml, která má poškozenou SIP hlavičku (hodnota pole Content-Length a skutečná velikost těla nesedí).

```
1 sipp 192.168.100.40 -sf sipp_uac_bad_message.xml
```

Kód 6.15: Příkaz nástroje SIPp pro testování poškozených SIP zpráv

V kódu 6.15 je zobrazen příkaz nástroje SIPp, který využívá XML šablonu. Pro využití šablon se používá parametr -sf. Na obrázku 6.16 je k vidění výpis Kamailio serveru do terminálu při



```
ERROR: {1 1 MESSAGE 1-3769@192.168.100.10} <script>: Malformed SIP request from 192.168.100.10:5060
INFO: <script>: route[HTTP_REPLY]: status 200
```

Obrázek 6.16: Ukázka výpisů Kamailio serveru během SIP komunikace s poškozenou SIP hlavičkou

zjištění, že je příchozí SIP zpráva poškozena. Na obrázku 6.17 lze vidět odpověď Kamailio serveru na příchozí poškozenou SIP zprávu. Jako důvod negativní odpověď Kamailio udává chybu 400 - Content-Length mis-match, což odpovídá použité šabloně. V příložených souborech k této práci je odchycená komunikace programem Wireshark. Popis struktury příloženého souboru je v příloze C.

```
received 'SIP/2.0 400 Content-Length mis-match'
Via: SIP/2.0/UDP 192.168.100.10:5060;branch=z9hG4bK-3769-1-0;rport=5060
From: sipp <sip:sipp@192.168.100.10:5060>;tag=3769SIPpTag001
To: sut <sip:service@192.168.100.40:5060>;tag=19d3aa08e358f57f78a0aea3b4cb3019.8c0470a4
Call-ID: 1-3769@192.168.100.10
CSeq: 1 MESSAGE
Server: kamailio (5.4.3 (x86_64/linux))
Content-Length: 0
```

Obrázek 6.17: Ukázka SIP odpovědi Kamailio serveru na poškozenou SIP zprávu

6.4.9 Pokus o nepovolenou registraci účastníka

Pomocí modulu Permissions, který je teoreticky probrán v 3.3.2, mohou být konkrétní IP adresy, nebo celé podsítě, vkládány do skupin. Nad těmito skupinami lze provádět různé kontroly. V práci je demonstrován případ registrace účastníků. Možnosti skupin jsou však velké, mohly by sloužit např. pro definici důvěryhodných serverů poskytovatelů služeb či okolních serverů v rámci podnikové infrastruktury. Pro účely testování byla v rámci Kamailio serveru vytvořena skupina číslo 200, pod kterou spadá podsít 192.168.100.0/25. Hostům z této podsítě je povolena registrace. Pro zbytek domácí sítě, tedy podsít 192.168.100.128/25, registrace fungovat nebude a informace o registrační pokusy budou zaslány na Ryu kontrolér.

```
1  if (method=="REGISTER") {
2      if (allow_source_address("200")) {
3          route(REGISTRAR);
4          xlog("Register from (IP:$si:$sp).");
5          exit;
6      }
7      sl_reply("401", "Address not authorized");
8      $http_req(hdr) = $null;
9      $http_req(method) = "POST";
10     $http_req(hdr) = "Content-type: Application/json";
11     $http_req(body) = "{\"nw_src\": \"\"+$si+\"/32\", \"actions\": \"REGISTER\",
        \"priority\": \"10\"}";
12     xlog("BAD REGISTER: Sending source IP $si:$sp address to Ryu controller.");
13     http_async_query("192.168.100.20:8080/firewall/rules/all", "HTTP_REPLY");
14     exit;
15 }
```

Kód 6.16: Příkaz nástroje SIPp pro testování poškozených SIP zpráv

Použití skupiny č. 200 uvnitř těla funkce modulu Permissions je vidět na řádce 2 kódu 6.16. Pokud zdrojová IP adresa REGISTER zprávy souhlasí s údaji ve skupině 200, provedou se řádky 3 až 5.

Na řádce 3 je použití Kamailio metody REGISTRAR, která se stará o registraci a přihlašování k účtům. Pokud REGISTER zpráva přijde z IP, která se skupinou č. 200 nesouhlasí, provede se kód na řádcích 7 až 14, tedy zaslání negativní odpovědi 401 - Address not authorized, a zaslání informace na Ryu kontrolér. Na obrázku 6.18 je k vidění výpis Kamailio serveru do terminálu při pokusu o

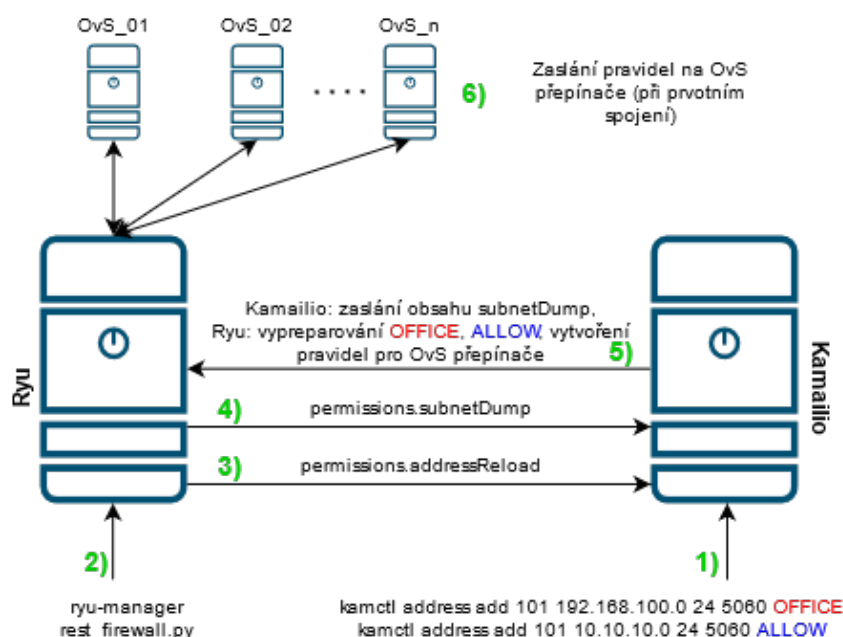
```
ERROR: {1 726 REGISTER zdxxxupfoqgpggrm@PC1} <script>: BAD REGISTER: Sending source IP 192.168.100.210:5060 address to Ryu controller.  
INFO: <script>: route[HTTP_REPLY]: status 200
```

Obrázek 6.18: Ukázka výpisu Kamailio serveru během registrace ze špatné IP adresy

registraci z nepovolené IP adresy. V příložených souborech k této práci je odchycená komunikace programem Wireshark. Popis struktury příloženého souboru je v příloze C.

6.4.10 Počáteční nastavení sítě skrze nástroj kamctl

Mimo identifikaci bezpečnostních hrozeb byla v práci implementována funkce nastavení sítě na Kamailio serveru skrze nástroj kamctl, které pak budou skrze Ryu kontrolér propagovány do všech nově připojených OvS přepínačů. Pomocí naimplementovaných metod a principů je možno definovat hosty a podsítě. Ti se budou vůči podnikové síti povolovat nebo blokovat. Následující ukázka je kvůli velikosti obrázku zkrácena pouze na zadefinování podnikové sítě a jedné podsítě, která má být povolena. Na obrázku 6.19 je znázorněna funkce nastavení sítě. Zelená čísla znázorňují pořadí



Obrázek 6.19: Počáteční nastavení sítě skrze nástroj kamctl

postupu. Jako první je pomocí Kamailio nástroje kamctl (viz 3.1) volán RPC příkaz modulu Permissions (viz 3.3.2) a zadefinují se dvě podsítě. Příkaz je složen z klíčových slov address a add, které

implikují, že se bude přidávat adresa nebo podsít, a řady parametrů - skupina, adresa, maska, port a tag. V práci je využit parametr tag k určení typu sítě. Podniková síť se označuje jako OFFICE. Síť, která bude moci s podnikovou sítí komunikovat, se označuje jako ALLOW. A síť, jejíž komunikace bude blokována, se označuje jako DENY. Nastavení blokování sítě by dávalo smysl v případě, že by byl firewall nastaven k povolení veškerého provozu, a blokování by byli jen útočníci. Druhá možnost využití je, kdyby byla povolena velká síť (např. s maskou /16), a z této podsítě by bylo potřeba vypreparovat a blokovat menší podsít (např. s maskou /28). Ve druhém kroce se zapne Ryu kontrolér s aplikací `rest_firewall.py`. Po startu aplikace pošle Ryu JSONRPC příkaz `addressReload` (viz řádky 1 až 5 v 6.17) skrze protokol HTTP na URL `http://192.168.100.40:8081/rpc`, kde má Kamailio otevřené spojení a vyčkává na příchod těchto příkazů. Jelikož je nebezpečné nechávat takové rozhraní bez restrikce, byl přístup omezen na IP adresy, které jsou přidány do skupiny 100 modulu `Permissions` (viz 3.3.8). Ve čtvrtém kroce zavolá metoda z řádků 1 až 5 v 6.17. Zde se po Kamailiu požaduje seznam definovaných podsítí. V pátém kroce Kamailio zašle Ryu seznam podsítí. Následně Ryu tento seznam zpracuje a rozpošle na každý nově připojený OvS přepínač pomocí kódu 6.19. K vypreparování podnikové sítě slouží funkce z řádků 7 až 12 v 6.17.

```
1 def kamailio_subnet_dump(self):
2     data = {'jsonrpc': '2.0', 'method': 'permissions.subnetDump'}
3     r = requests.get('http://192.168.100.40:8081/rpc', json=data).json()
4     message = r["result"]
5     return message
6
7 def kamailio_address_reload(self):
8     data = {'jsonrpc': '2.0', 'method': 'permissions.addressReload'}
9     r = requests.get('http://192.168.100.40:8081/rpc', json=data).json()
10    message = r["result"]
11    print(message)
12
13 def officeParsing(message):
14     office = ""
15     for key in message:
16         if key["item"]["tag"] == "OFFICE":
17             office = str(key["item"]["ip"]) + "/" + str(key["item"]["mask"])
18     return office
```

Kód 6.17: Python metody pro `Permissions` modul

Kód 6.19 se volá z metody `regist_ofs` třídy `FirewallController`. Před tím, než je tímto kódem zahájena komunikace s Kamailio serverem, se provádí kód 6.18. Řádky 1 a 2 slouží k uložení statických IP adres Ryu kontroléru a Kamailio serveru do proměnných. Na řádcích 3 a 4 je definice dvou Python

slovníků, které jsou předávány jako argument metodě `set_rule` na řádcích 9 a 10. V rámci slovníků je nadefinován záznam pro Flow tabulku, který povoluje komunikaci mezi kontrolérem a Kamailio serverem. Řádek 5 slouží k přidání právě registrovaného OpenFlow přepínače do seznamu třídy `FirewallController`. Řádek 6 slouží k vytvoření záznamu, který povoluje ARP protokol. Řádek 7 slouží k vypnutí zasílání stavu přepínače na Ryu kontrolér. Při záměně slova `disable` za `enable` se zasílání opět zapne, je však velmi nepřehledné. Řádek 8 povoluje zpracovávání síťové komunikace spravovaným přepínačem. Originální `rest_firewall` aplikace má zasílání stavů zapnuto a zpracování síťové komunikace vypnuto. Toto nastavení lze ovlivňovat skrze REST rozhraní, např. pomocí nástroje `cURL`.

```
1 Ryu_IP = "192.168.100.20"
2 Kamailio_IP = "192.168.100.40"
3 initial_networking1 = {'nw_src': Ryu_IP, 'nw_dst': Kamailio_IP, 'actions': '
    ALLOW', 'priority': '1000'}
4 initial_networking2 = {'nw_src': Kamailio_IP, 'nw_dst': Ryu_IP, 'actions': '
    ALLOW', 'priority': '1000'}
5 FirewallController._OFS_LIST.setdefault(dp.id, f_ofs)
6 f_ofs.set_arp_flow()
7 f_ofs.set_log_disable()
8 f_ofs.set_enable_flow()
9 f_ofs.set_rule(initial_networking1, {}, 0)
10 f_ofs.set_rule(initial_networking2, {}, 0)
```

Kód 6.18: Začátek metody `regist_ofs`

Na prvních třech řádcích kódu 6.19 se volají metody popsané v 6.17. Do proměnné `office` se vypíše síť s tagem `OFFICE`, do proměnné `subnets` se z volání funkce vrátí seznam všech definovaných sítí. Řádky 5 až 12, resp. 13 až 19 pak slouží k promíchání podnikové sítě se sítěmi, které mají v parametru tag `ALLOW`, resp. `DENY`, a to skrze smyčku `for`. Vždy je vytvořena dvojice dvojice slovníků - síť A na síť B, resp. síť B na síť A. Pokud je povolen jen jeden směr, síťový provoz přes firewall neprojde. Slovníky se předávají do metody `set_rule`. Pomocí posledního řádku se do terminálu vypíše hláška, která je k vidění na obrázku A.3.

```
1 f_ofs.kamailio_address_reload()
2 subnets = f_ofs.kamailio_subnet_dump()
3 office = FirewallController.officeParsing(subnets)
4 print("Kamailio - office network:", office)
5 for key in subnets:
6     if key["item"]["tag"] == "ALLOW":
7         trusted_subnet = str(key["item"]["ip"]) + "/" + str(key["item"]["mask"]
8                                     ])
```

```

8      dict1 = {'nw_src': office, 'nw_dst': trusted_subnet, 'actions': 'ALLOW
          ', 'priority': '5'}
9      dict2 = {'nw_src': trusted_subnet, 'nw_dst': office, 'actions': 'ALLOW
          ', 'priority': '5'}
10     f_ofs.set_rule(dict1, {}, 0)
11     f_ofs.set_rule(dict2, {}, 0)
12     print("Kamailio - allow network:", trusted_subnet)
13     elif key["item"]["tag"] == "DENY":
14         drop_subnet = str(key["item"]["ip"]) + "/" + str(key["item"]["mask"])
15         dict1 = {'nw_src': office, 'nw_dst': drop_subnet, 'actions': 'DENY', '
            priority': '5'}
16         dict2 = {'nw_src': drop_subnet, 'nw_dst': office, 'actions': 'DENY', '
            priority': '5'}
17         f_ofs.set_rule(dict1, {}, 0)
18         f_ofs.set_rule(dict2, {}, 0)
19         print("Kamailio - deny network:", drop_subnet)
20 FirewallController._LOGGER.info('dpid=%s: Join as firewall.', dpid_str)

```

Kód 6.19: Zpracování definovaných sítí a zaslání na OvS přepínač

Procedure popsaná v této sekci je prováděna vždy, když se ke kontroléru připojuje nový OpenFlow přepínač. Tím je zaručeno, že budou mít všechny přepínače sesynchronizovány záznamy ve Flow tabulce. To ovšem platí pouze do doby, než někdo změní seznam sítí v Kamailiu po tom, co už se nějaký přepínač zaregistroval. Na obrázku A.1 v příloze A je porovnání volání příkazu subnetDump vzdáleně skrze JSONRPC a HTTP protokol, a lokálně s použitím nástroje kamcmd.

6.5 Shrnutí navrženého řešení

Navržené řešení aplikačního firewallu funguje jako spolupráce tří prvků - Kamailio SIP serveru, SDN kontroléru Ryu a virtuálního přepínače Open vSwitch. Tyto prvky byly nainstalovány na fyzických počítačích, které jsou součástí domácí sítě. K testování firewallu je v síti přítomen počítač s operačním systémem Kali Linux. Komunikace mezi Kamailio SIP serverem a Ryu kontrolérem probíhá pomocí HTTP zpráv, jež mají strukturované tělo do JSON formátu. Komunikace mezi Ryu kontrolérem a Open vSwitch přepínačem probíhá pomocí protokolu OpenFlow.

Kamailio server je v práci využit co by detektor bezpečnostních hrozeb. Při odhalení hrozby zasílá informace na kontrolér. Kontrolér využívá vestavěnou aplikaci, která umí vytvářet záznamy ve spravovaných OpenFlow zařízeních. Aplikace nabízí možnost vytváření záznamů, které budou blokovat síťový provoz z konkrétních IP adres. Open vSwitch přepínač tak může zastát roli firewallu, který

blokuje provoz na základě instrukcí, které mu předá Ryu kontrolér.

Vestavěná Ryu aplikace byla patřičně upravena k tomu, aby z příchozích HTTP zpráv dokázala rozpoznat typy hrozeb. Kamailio vždy zasílá zdrojovou IP adresu útočníka a typ hrozby. V rámci aplikace se vytvářejí záznamy v lokální databázi, které k jednotlivým IP adresám útočníků udržují počítadla jednotlivých typů hrozeb, časové razítko a životnost záznamu. Kamailio umí rozpoznat 7 typů hrozeb - útok záplavou SIP žádostí (flood), SQL injektování (sql), skenování sítě (scanner), spam ve VoIP sítích (spam), poškozené/pozměněné SIP zprávy (malformed), SIP zprávy z nepovolených států (geoip2) a pokus o registraci ze špatné IP (register). V aplikaci je implementována logika počítání doby, po kterou bude blokující záznam v přepínači přítomen. Výpočet této životnosti je závislý na počítadlech, které se udržují v lokální databázi. Aplikace také umí záznamy z databáze vymazat, pokud jsou delší dobu neaktivní. Vzhledem k implementaci logiky časových razítek a životností je lokální databáze do jisté míry synchronizována s Flow tabulkou přepínačů.

Mimo odhalování hrozeb byla v rámci firewallu implementována funkce nastavení sítě pro každý nově připojený OpenFlow přepínač. Na Kamailio serveru lze skrze nástroj kamctl definovat podnikovou síť, a k ní příslušné sítě, které se mají při registraci přepínače ihned povolit, nebo zakázat. Aplikace se na tyto údaje doptává pomocí JSONRPC příkazů, které jsou na Kamailio posílány pomocí protokolu HTTP.

Oproti tradičnímu přístupu má výsledné řešení hned několik výhod. Použité prvky sítě jsou open-source, a volně k dispozici. Vzhledem k využití protokolu OpenFlow ke správě zařízení, které slouží k blokaci síťového provozu, je řešení snadno rozšiřitelné o další blokující prvky. Další výhodou je volba využití frameworku Ryu, jelikož jsou jeho aplikace psány v jazyce Python. Tento jazyk má velké množství balíčků, vývojářů a možností, jak tyto aplikace propojovat s dalšími systémy, jak utvářet komplexní analýzy nad shromážděnými daty atp. Jelikož se informace o hrozbách zasílají skrze protokol HTTP, a útočníci jsou definováni IP adresou a klíčovým slovem hrozby, je firewall snadno rozšiřitelný jak o odhalování dalších typů hrozeb, tak i o servery, které odahlování provádějí. Tělo HTTP zprávy je navrženo tak, aby se kontrola hrozeb mohla provádět i nad jinými protokoly, než je SIP. Poslední výhoda je v zabezpečení, kdy protokoly HTTP i OpenFlow dokážou být zabezpečeny pomocí TLS.

Hlavní nevýhodou navrženého řešení je použití Open vSwitch přepínače ve stolních počítačích. Open vSwitch sice využívá Kernel modul k urychlení přepínání paketů, rychlostí se však nevyrovná integrovaným ASIC obvodům, které jsou k nalezení v tradičních firewallech. Tento nedostatek je však minimální, jelikož může být ke kontrolé připojeno jakékoliv zařízení, které je schopno být spravováno OpenFlow protokolem. Mezi taková zařízení patří např. Cisco Nexus 9000, Juniper QFX5100 nebo Dell S6000. Všechna tato zařízení podporují protokol OpenFlow verze 1.3.

Kapitola 7

Závěr

Hlavním cílem diplomové práce bylo využití Kamailio SIP serveru a SDN infrastruktury k návrhu řešení, které bude schopno využívat SIP server co by detektor hrozeb, a OpenFlow přepínač co by firewall. Vzhledem k povaze tohoto zadání bylo také nutné vybrat vhodný SDN kontrolér.

K dosažení výsledného řešení bylo nezbytné nastudovat problematiku protokolů SIP a OpenFlow, Kamailio SIP serveru, SDN paradigmatu a Open vSwitch virtuálního přepínače. V rámci teoretické části práce byl kladen důraz na seznámení se s Kamailio moduly, které slouží k detekci bezpečnostních hrozeb. Dále bylo nutné osvojit si práci s SDN kontrolérem Ryu, který je v návrhu řešení využit.

Výsledné řešení je spojením tří prvků - Kamailio SIP serveru, SDN kontroléru Ryu a virtuálního přepínače Open vSwitch. Vhodná volba komunikačních procedur, protokolů a SDN aplikace však umožňuje bezproblémové přidávání dalších OpenFlow přepínačů do již existující architektury sítě. Návrh řešení má povahu příkladu a konfigurace a implementace jsou vedeny formou ukázek. Řešení však obsahuje open-source prvky a k diplomové práci jsou přiloženy okomentované zdrojové kódy, Firewall tak může být dále rozvíjen a obohacován.

Přínosem diplomové práce je ukázka propojení SDN infrastruktury se SIP serverem, který slouží jako detektor vybraných bezpečnostních hrozeb. Výsledné řešení může sloužit jako základ pro diplomovou práci, jejíž úkol by byl Firewall zdokonalit. Mezi návrhy vylepšení patří zabezpečení protokolů HTTP a OpenFlow, hlubší logika odhalování některých typů hrozeb (skenování sítí, VoIP spam) a zvýšení inteligence výpočtu životnosti záznamů pro Flow tabulku.

Literatura

- [1] HANDLEY, M., H. SCHULZRINNE, E. SCHOOLER a J. ROSENBERG. SIP: Session Initiation Protocol [online]. Request for Comments: 3261, Červen 2002. Dostupné z: <https://tools.ietf.org/html/rfc3261>
- [2] CABIDDU, Federico, Giacomo VACCA a Camille OUDOT. HTTP_ASYNC_CLIENT Module [online]. Kamailio, 5.4.x. Dostupné z: https://www.kamailio.net/docs/modules/stable/modules/http_async_client.html
- [3] IANCU, Bogdan-Andrei. PIKE Module [online]. Kamailio, 5.4.x. Dostupné z: <https://www.kamailio.net/docs/modules/stable/modules/pike.html>
- [4] TIRPAK, Miklos. permissions Module [online]. Kamailio, 5.4.x. Dostupné z: <https://www.kamailio.net/docs/modules/stable/modules/permissions.html>
- [5] OHLMEIER, Nils. The Sanity Module - SIP syntax checking [online]. Kamailio, 5.4.x. Dostupné z: <https://www.kamailio.net/docs/modules/stable/modules/sanity.html>
- [6] MODROIU, Elena-Ramona. HTable Module [online]. Kamailio, 5.4.x. Dostupné z: <https://www.kamailio.net/docs/modules/stable/modules/htable.html>
- [7] OKHAPKIN, Sergey. geoip2 Module [online]. Kamailio, 5.4.x. Dostupné z: <https://www.kamailio.net/docs/modules/stable/modules/geoip2.html>
- [8] IANCU, Bogdan-Andrei. dialog Module [online]. Kamailio, 5.4.x. Dostupné z: <https://www.kamailio.net/docs/modules/stable/modules/dialog.html>
- [9] MIERLA, Daniel-Constantin. JSONRPC-S (jsonrpc server) Module [online]. Kamailio, 5.4.x. Dostupné z: <https://www.kamailio.net/docs/modules/stable/modules/jsonrpcs.html>
- [10] How does a hash table work? Stack Overflow [online]. Bentima House, London, 2009. Dostupné z: <https://stackoverflow.com/questions/730620/how-does-a-hash-table-work>
- [11] GeoLite2 Free Geolocation Data. MaxMind [online]. Waltham, USA. Dostupné z: <https://dev.maxmind.com/geoip/geoip2/geolite2/>

- [12] Pseudo-Variables. Kamailio SIP Server WIKI [online]. Kamailio SIP Server v5.4.x (stable). Dostupné z: <https://www.kamailio.org/wiki/cookbooks/5.4.x/pseudovariables>
- [13] Kamailio - Getting Started Guide. Kamailio SIP Server Wiki [online]. dokuwiki. Dostupné z: <https://www.kamailio.org/wiki/tutorials/getting-started/main>
- [14] Kamailio SIP Server Documentation Wiki. Kamailio SIP Server Wiki [online]. dokuwiki. Dostupné z: <https://www.kamailio.org/wiki/start>
- [15] Welcome To Kamailio – The Open Source SIP Server. Kamailio [online]. Dostupné z: <https://www.kamailio.org/w/>
- [16] Overview of Security related config snippets. Kamailio SIP Server Wiki [online]. dokuwiki. Dostupné z: <https://www.kamailio.org/wiki/tutorials/security/kamailio-security>
- [17] OpenFlow. Flowgrammable [online]. Dostupné z: <http://flowgrammable.org/sdn/openflow/>
- [18] NADEAU, Thomas D. a Ken GRAY. SDN: Software Defined Networks [online]. 1. USA: O'Reilly, 2013 [cit. 2021-04-04]. ISBN 978-1-449-34230-2. Dostupné z: <https://www.oreilly.com/library/view/sdn-software-defined/9781449342425/>
- [19] FERNANDEZ, Carlos a Jose L. MUÑOZ. Software Defined Networking (SDN) with OpenFlow 1.3, Open vSwitch and Ryu [online]. Universitat Politècnica de Catalunya, Telematics Department, Bachelor thesis, 2015. Dostupné z: <https://upcommons.upc.edu/handle/2117/77684>
- [20] OpenDaylight User Guide. OpenDaylight Project [online]. Dostupné z: <https://nexus.opendaylight.org/content/sites/site/org.opendaylight/docs/master/userguide/manuals/userguide/bk-user-guide/content/index.html>
- [21] Getting Started Guide - OpenDaylight Documentation. [online]. Dostupné z: <https://docs.opendaylight.org/en/latest/getting-started-guide/index.html>
- [22] Open Network Operating System. ONOS - Wiki [online]. Dostupné z: <https://wiki.onosproject.org/display/ONOS/ONOS>
- [23] Overview of ONOS architecture. ONOS - Wiki [online]. Dostupné z: <https://wiki.onosproject.org/display/ONOS/Overview+of+ONOS+architecture>
- [24] VANNEST, Michael. ONOS and ODL: Know the difference [online]. Rochester, NY: CloudSmartz, 2016. Dostupné z: <https://cloudsmartz.com/insights/onos-and-odl-know-the-difference/>
- [25] Welcome to RYU the Network Operating System(NOS) [online]. Read the Docs, 2014. Dostupné z: <https://ryu.readthedocs.io/en/latest/index.html>

- [26] Architecture - Ryubook 1.0 documentation [online]. GitHub Docs. Dostupné z: <https://osrg.github.io/ryu-book/en/html/arch.html>
- [27] Firewall - Ryubook 1.0 documentation [online]. GitHub Docs. Dostupné z: https://osrg.github.io/ryu-book/en/html/rest_firewall.html
- [28] CERON, João, Klaus STEDING-JESSEN a Cristine HOEPERS. Anatomy of SIP Attacks [online]. USENIX, 2012. Dostupné z: https://www.usenix.org/system/files/login/articles/login1212_ceron.pdf
- [29] Open vSwitch: Production Quality, Multilayer Open Virtual Switch [online]. Linux Foundation Collaborative Projects. Dostupné z: <https://www.openvswitch.org/>
- [30] MENDIOLA, Alaitz, Eduardo JACOB, Jasone ASTORGA a Marivi HIGUERO. A Survey on the Contributions of Software-Defined Networking to Traffic Engineering. IEEE Communications Surveys & Tutorials [online]. ResearchGate, Květen 2017. Dostupné z: doi:10.1109/COMST.2016.2633579
- [31] Open vSwitch 2.15.90 Manpages [online]. Dostupné z: <https://www.openvswitch.org/support/dist-docs/>
- [32] OpenFlow Switch Specification: Version 1.3.0 [online]. Open Networking Foundation, 2012. Dostupné z: <https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf>

Příloha A

Obrázky

```
GET /rpc HTTP/1.1
Host: 192.168.100.40:8081
User-Agent: python-requests/2.22.0
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
Content-Length: 54
Content-Type: application/json

{"jsonrpc": "2.0", "method": "permissions.subnetDump"} HTTP/1.1 200 OK
Sia: SIP/2.0/TCP 192.168.100.20:42488
Content-Type: application/json
Server: kamailio (5.4.3 (x86_64/linux))
Content-Length: 446

{
  "jsonrpc": "2.0",
  "result": [
    {
      "id": 0,
      "group": 101,
      "item": {
        "ip": "192.168.100.0",
        "mask": 24,
        "port": 5060,
        "tag": "OFFICE"
      }
    },
    {
      "id": 1,
      "group": 101,
      "item": {
        "ip": "192.168.100.0",
        "mask": 24,
        "port": 5060,
        "tag": "ALLOW"
      }
    },
    {
      "id": 2,
      "group": 200,
      "item": {
        "ip": "192.168.100.0",
        "mask": 25,
        "port": 5060,
        "tag": "register_trust"
      }
    }
  ]
}
```

(a) Volání příkazu subnetDump vzdáleně skrze JSONRPC a HTTP protokol

```
root@PC4:~# kamcmd permissions.subnetDump
{
  id: 0
  group: 101
  item: {
    ip: 192.168.100.0
    mask: 24
    port: 5060
    tag: OFFICE
  }
}
{
  id: 1
  group: 101
  item: {
    ip: 192.168.100.0
    mask: 24
    port: 5060
    tag: ALLOW
  }
}
{
  id: 2
  group: 200
  item: {
    ip: 192.168.100.0
    mask: 25
    port: 5060
    tag: register_trust
  }
}
root@PC4:~#
```

(b) Volání příkazu subnetDump lokálně nástrojem kamcmd

Obrázek A.1: Porovnání volání příkazu subnetDump vzdáleně a lokálně

No.	Time	Source	Destination	Protocol	Length	Info	No.	Time	Source	Destination	Protocol	Length	Info
3	0.011042038	192.168.100.30	192.168.100.20	OpenFlow	74	Type: OFPT_ECHO_REQUEST	3	0.011042038	192.168.100.30	192.168.100.20	OpenFlow	74	Type: OFPT_ECHO_REQUEST
4	0.012335960	192.168.100.20	192.168.100.30	OpenFlow	74	Type: OFPT_ECHO_REPLY	4	0.012335960	192.168.100.20	192.168.100.30	OpenFlow	74	Type: OFPT_ECHO_REPLY
5	0.013136753	192.168.100.30	192.168.100.20	OpenFlow	74	Type: OFPT_ECHO_REQUEST	5	0.013136753	192.168.100.30	192.168.100.20	OpenFlow	74	Type: OFPT_ECHO_REQUEST
11	5.014728552	192.168.100.20	192.168.100.30	OpenFlow	74	Type: OFPT_ECHO_REPLY	11	5.014728552	192.168.100.20	192.168.100.30	OpenFlow	74	Type: OFPT_ECHO_REPLY
19	10.014781664	192.168.100.30	192.168.100.20	OpenFlow	74	Type: OFPT_ECHO_REQUEST	19	10.014781664	192.168.100.30	192.168.100.20	OpenFlow	74	Type: OFPT_ECHO_REQUEST
29	19.016249032	192.168.100.20	192.168.100.30	OpenFlow	74	Type: OFPT_ECHO_REPLY	29	19.016249032	192.168.100.20	192.168.100.30	OpenFlow	74	Type: OFPT_ECHO_REPLY
29	15.017623859	192.168.100.30	192.168.100.20	OpenFlow	74	Type: OFPT_ECHO_REQUEST	29	15.017623859	192.168.100.30	192.168.100.20	OpenFlow	74	Type: OFPT_ECHO_REQUEST
30	15.018176431	192.168.100.20	192.168.100.30	OpenFlow	74	Type: OFPT_ECHO_REPLY	30	15.018176431	192.168.100.20	192.168.100.30	OpenFlow	74	Type: OFPT_ECHO_REPLY

Frame 3: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface bridge, id 0

Ethernet II, Src: SpeedDra_9c:88:dd (00:13:3b:9c:88:dd), Dst: ASUSTekC_26:9a:8a (60:45:cb:26:9a:8a)

Internet Protocol Version 4, Src: 192.168.100.30, Dst: 192.168.100.20

Transmission Control Protocol, Src Port: 59064, Dst Port: 6633, Seq: 1, Ack: 1, Len: 8

OpenFlow 1.3

Version: 1.3 (0x04)

Type: OFPT_ECHO_REQUEST (2)

Length: 8

Transaction ID: 0

Frame 4: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface bridge, id 0

Ethernet II, Src: ASUSTekC_26:9a:8a (60:45:cb:26:9a:8a), Dst: SpeedDra_9c:88:dd (00:13:3b:9c:88:dd)

Internet Protocol Version 4, Src: 192.168.100.20, Dst: 192.168.100.30

Transmission Control Protocol, Src Port: 6633, Dst Port: 59064, Seq: 1, Ack: 0, Len: 8

OpenFlow 1.3

Version: 1.3 (0x04)

Type: OFPT_ECHO_REPLY (3)

Length: 8

Transaction ID: 0

(a) OpenFlow Echo Request

(b) OpenFlow Echo Reply

Obrázek A.2: OpenFlow Keepalive zprávy

```

root@PC2:~# systemctl status restfirewall.service
● restfirewall.service - REST Firewall application for Ryu controller
   Loaded: loaded (/lib/systemd/system/restfirewall.service; enabled; vendor preset: enabled)
   Active: active (running) since Sun 2021-04-25 17:43:01 CEST; 9s ago
     Main PID: 3677 (ryu-manager)
        Tasks: 1 (limit: 9303)
       Memory: 42.9M
      CGroup: /system.slice/restfirewall.service
              └─3677 /usr/bin/python3 /usr/local/bin/ryu-manager /root/rest_firewall.py

Apr 25 17:43:02 PC2 ryu-manager[3677]: Generating grammar tables from /usr/lib/python3.8/lib2to3/Grammar.txt
Apr 25 17:43:02 PC2 ryu-manager[3677]: Generating grammar tables from /usr/lib/python3.8/lib2to3/PatternGrammar.txt
Apr 25 17:43:02 PC2 ryu-manager[3677]: loading app ryu.controller.ofp_handler
Apr 25 17:43:02 PC2 ryu-manager[3677]: instantiating app None of DPSet
Apr 25 17:43:02 PC2 ryu-manager[3677]: creating context dpset
Apr 25 17:43:02 PC2 ryu-manager[3677]: creating context wsgi
Apr 25 17:43:02 PC2 ryu-manager[3677]: instantiating app /root/rest_firewall.py of RestFirewallAPI
Apr 25 17:43:02 PC2 ryu-manager[3677]: instantiating app ryu.controller.ofp_handler of OFPHandler
Apr 25 17:43:02 PC2 ryu-manager[3677]: (3677) wsgi starting up on http://0.0.0.0:8080
Apr 25 17:43:09 PC2 ryu-manager[3677]: [FW][INFO] dpid=000000133b9c88dd: Join as firewall.

```

Obrázek A.3: Spouštění Ryu aplikace jako službu v systemd

Database record with IP: 192.168.100.190/32 was created.
192.168.100.40 - - [18/Apr/2021 19:13:37] "POST /firewall/rules/all HTTP/1.1" 200 226 0.041123
Database record with IP: 192.168.100.191/32 was created.
192.168.100.40 - - [18/Apr/2021 19:13:37] "POST /firewall/rules/all HTTP/1.1" 200 226 0.020189
Database record with IP: 192.168.100.192/32 was created.
192.168.100.40 - - [18/Apr/2021 19:13:37] "POST /firewall/rules/all HTTP/1.1" 200 226 0.030052
Database record with IP: 192.168.100.193/32 was created.
192.168.100.40 - - [18/Apr/2021 19:13:37] "POST /firewall/rules/all HTTP/1.1" 200 226 0.029548
Database record with IP: 192.168.100.194/32 was created.
192.168.100.40 - - [18/Apr/2021 19:13:37] "POST /firewall/rules/all HTTP/1.1" 200 227 0.030004
Database record with IP: 192.168.100.195/32 was created.
192.168.100.40 - - [18/Apr/2021 19:13:37] "POST /firewall/rules/all HTTP/1.1" 200 227 0.029300
Database record with IP: 192.168.100.196/32 was created.
192.168.100.40 - - [18/Apr/2021 19:13:37] "POST /firewall/rules/all HTTP/1.1" 200 227 0.026988
Database record with IP: 192.168.100.197/32 was created.
192.168.100.40 - - [18/Apr/2021 19:13:38] "POST /firewall/rules/all HTTP/1.1" 200 227 0.047771
Database record with IP: 192.168.100.198/32 was created.
192.168.100.40 - - [18/Apr/2021 19:13:38] "POST /firewall/rules/all HTTP/1.1" 200 227 0.021086
Database record with IP: 192.168.100.199/32 was created.
192.168.100.40 - - [18/Apr/2021 19:13:38] "POST /firewall/rules/all HTTP/1.1" 200 227 0.026883
Database record with IP: 192.168.100.200/32 was created.
192.168.100.40 - - [18/Apr/2021 19:13:38] "POST /firewall/rules/all HTTP/1.1" 200 227 0.036776

Deleting inactive IP: 192.168.100.189/32 from database
Deleting inactive IP: 192.168.100.190/32 from database
Deleting inactive IP: 192.168.100.191/32 from database
Deleting inactive IP: 192.168.100.192/32 from database
Deleting inactive IP: 192.168.100.193/32 from database
Deleting inactive IP: 192.168.100.194/32 from database
Deleting inactive IP: 192.168.100.195/32 from database
Deleting inactive IP: 192.168.100.196/32 from database
Deleting inactive IP: 192.168.100.197/32 from database
Deleting inactive IP: 192.168.100.198/32 from database
Deleting inactive IP: 192.168.100.199/32 from database
Deleting inactive IP: 192.168.100.200/32 from database
127.0.0.1 - - [18/Apr/2021 19:17:56] "POST /firewall/rules/all HTTP/1.1" 200 227 0.004134

(a) Vytváření záznamů v databázi při útoku typu flood z více IP adres

(b) Mazání záznamů z databáze

Obrázek A.4: Vytváření a mazání záznamů z Ryu databáze

Příloha B

Kód

Tato příloha slouží pro uložení příliš dlouhého kusu kódu. V textu práce je na něj odkazováno. Kód je vyjmut z Ryu aplikace `rest_firewall.py`. Celá aplikace je k nalezení v přiložených souborech.

```
1 def which_action(self, argument):
2     timestamp = time.time()
3     action = argument["actions"]
4     action = action.casefold()
5     src_ip = argument["nw_src"]
6     if action == "scanner":
7         self.database_update(src_ip, timestamp, False)
8         argument["actions"] = "DENY"
9         self.database[src_ip]["scanner"] += 1
10        lifetime = self.action_processing(src_ip, timestamp)
11    elif action == "register":
12        self.database_update(src_ip, timestamp, False)
13        argument["actions"] = "DENY"
14        self.database[src_ip]["register"] += 1
15        lifetime = self.action_processing(src_ip, timestamp)
16    elif action == "flood":
17        self.database_update(src_ip, timestamp, True)
18        argument["actions"] = "DENY"
19        self.database[src_ip]["flood"] += 1
20        lifetime = self.action_processing(src_ip, timestamp)
21    elif action == "malformed":
22        self.database_update(src_ip, timestamp, False)
23        argument["actions"] = "DENY"
24        self.database[src_ip]["malformed"] += 1
```

```

25         lifetime = self.action_processing(src_ip, timestamp)
26     elif action == "geoip":
27         self.database_update(src_ip, timestamp, False)
28         argument["actions"] = "DENY"
29         self.database[src_ip]["geoip"] += 1
30         lifetime = self.action_processing(src_ip, timestamp)
31     elif action == "sql":
32         self.database_update(src_ip, timestamp, False)
33         argument["actions"] = "DENY"
34         self.database[src_ip]["sql"] += 1
35         lifetime = self.action_processing(src_ip, timestamp)
36     elif action == "spam":
37         self.database_update(src_ip, timestamp, False)
38         argument["actions"] = "DENY"
39         if (time.time() - self.database[src_ip]["timestamp"] < 180):
40             self.database[src_ip]["spam"] += 1
41         lifetime = self.action_processing(src_ip, timestamp)
42     else:
43         print('Keywords not found.')
44         lifetime = 0
45     return argument, lifetime
46
47 def database_update(self, src_ip, timestamp, skip_delete):
48     if src_ip not in self.database:
49         self.database[src_ip] = {"timestamp": timestamp, "lifetime": 20, "
50             flood": 0, "register": 0, "geoip": 0, "sql": 0, "malformed": 0, "
51             scanner": 0, "spam": 0}
52         print('Database record with IP:', src_ip, ' was created.')
53         if skip_delete == False:
54             inactive_IP = []
55             for IP in self.database:
56                 if (time.time() - self.database[IP]["timestamp"]) > 2*(self.
57                     database[src_ip]["lifetime"]):
58                     inactive_IP.append(IP)
59             for ip in inactive_IP:
60                 if not inactive_IP:
61                     print('All inactive IP addresses were deleted from database.
62                         ')

```

```

59         else:
60             print('Deleting inactive IP: ', ip, ' from database')
61             self.database.pop(ip)
62     else:
63         print('Timestamp of IP: ', src_ip, ' was updated.')
64         self.database[src_ip]["timestamp"] = timestamp
65
66     def action_processing(self, src_ip, timestamp):
67         rule_lifetime = self.database[src_ip]["lifetime"]
68         if (self.database[src_ip]["flood"] > 1):
69             rule_lifetime += 1
70         elif (self.database[src_ip]["register"] > 1):
71             rule_lifetime += 1
72         elif (self.database[src_ip]["sql"] > 1):
73             rule_lifetime += 1
74         elif (self.database[src_ip]["geoip"] > 1):
75             rule_lifetime += 1
76         elif (self.database[src_ip]["malformed"] > 1):
77             rule_lifetime += 1
78         elif (self.database[src_ip]["scanner"] > 1):
79             rule_lifetime += 1
80         elif (self.database[src_ip]["spam"] > 1):
81             rule_lifetime += 1
82         self.database[src_ip]["lifetime"] = rule_lifetime
83         return rule_lifetime
84
85     def _to_of_flow(self, cookie, priority, match, actions):
86         flow = {'cookie': cookie,
87                 'priority': priority,
88                 'flags': 0,
89                 'idle_timeout': 0,
90                 'hard_timeout': self.lifetime,
91                 'match': match,
92                 'actions': actions}
93         return flow

```

Kód B.1: Zpracování dat z Kamailio serveru v Ryu aplikaci

Příloha C

Struktura přiložených souborů

Tabulka C.1 slouží k orientaci v přiloženém souboru k diplomové práci. První složka obsahuje konfigurační soubor Kamailio SIP serveru a aplikaci pro Ryu kontrolér. Zbýlých osm složek představuje zachycenou komunikaci programem Wireshark (s příponou .pcapng). Soubory jsou pojmenovány podle místa, kde byl program Wireshark spouštěn. Složka *Komunikace s pákistánskou IP* obsahuje dva soubory Kamailio, jelikož byl provoz zachytáván na fyzickém (KamailioHTTP) a localhost (KamailioSIP) rozhraní. Kamailio soubory obsahují SIP a HTTP komunikaci, Ryu soubory pak HTTP a OpenFlow komunikaci. Pro snadnější orientaci lze využít filtr *sip or http*, resp. *http or openflow*.

Název složky	Obsah složky	Příslušná sekce v textu
Kódy	kamailio.cfg rest_firewall.py	
Komunikace s pákistánskou IP	KamailioHTTP KamailioSIP Ryu	6.4.7 SIP zpráva z nepovoleného státu
Nepovolená registrace	Kamailio Ryu	6.4.9 Pokus o nepovolenou registraci účastníka
Počáteční nastavení sítě	Kamailio Ryu	6.4.10 Počáteční nastavení sítě skrze nástroj kamctl
Poškozená SIP hlavička	Kamailio Ryu	6.4.8 Odhalení pozměněné/poškozené SIP zprávy
Skenování sítě	Kamailio Ryu	6.4.5 Skenování sítě
SQL injektování	Kamailio Ryu	6.4.4 Útok typu SQL injekce
Útok typu flood	Kamailio Ryu	6.4.3 Útok typu flood
Velmi krátký hovor	Kamailio Ryu	6.4.6 VoIP spam

Tabulka C.1: Struktura přiloženého souboru k diplomové práci